4

Massachusetts Institute of Technology

Laboratory for Computer Science

Progress Report 23

July 1985-
June 1986

Massachusetts Institute of Technology
Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139
617-253-5851

The work reported herein was carried out within the Laboratory for Computer Science, an MIT interdepartmental laboratory. During 1985-86 the principal financial support of the Laboratory has come from the Defense Advanced Research Projects Agency (DARPA). DARPA has been instrumental in supporting most of our research during the last 23 years and is gratefully acknowledged here. Our overall support has come from the following organizations:

- Defense Advanced Research Projects Agency;
- Department of Energy;
- National Institutes of Health;
- National Science Foundation;
- Office of Naval Research;
- United States Air Force;
- United States Army Research Office;
- MIT controlled IBM funds under an IBM/MIT joint study contract.

Other support of a generally smaller level has come from Harris Corporation, Siemens, and Giers.

Final assembly and production of this report was done by Paula Vancini with special assistance from Maria Sensale and Mary Weston.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/PR - 23	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION MIT Lab for Computer Science 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Dept. of Navy
6c. ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7b. ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

MIT Laboratory for Computer Science Progress Report 23

12. PERSONAL AUTHOR(S)
Dertouzos, M.L.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical/Progress	FROM 7/85 TO 6/86	June 1986	292

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Computer Architecture, Computer Science, Computer Systems, Electrical Engineering, Networks Theory of Computers, Programming Languages

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Annual Report of Progress made at the MIT Laboratory for Computer Science Under contracts:

a.) N00014-83k-0125,Darpa Order 5602/2095
b.) N00014-84k-0059, Darpa Order 4920

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Carol Nicolora	22b. TELEPHONE (Include Area Code) (617) 253-5894 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

★U.S. Government Printing Office: 1985-507-047

A

Contents:

B

c

ADMINISTRATION

Academic Staff

M. Dertouzos	Director
R. Rivest	Associate Director

Administrative Staff

P. Anderegg	Assistant Administrative Officer
A. Chow	Fiscal Officer
G. Brown	Facilities Officer
J. Hynes	Administrative Officer
M. Jones	Assistant Administrative Officer
M. Sensale	Librarian

Support Staff

L. Cavallaro	B. Pierce
R. Donahue	E. Profirio
M. Gibson	D. Simmons
A. Kekejian	P. Vancini
T. LoDuca	S. Van Norden

INTRODUCTION

The MIT Laboratory for Computer Science (LCS) is an interdepartmental laboratory whose principal goal is research in computer science and engineering.

Founded in 1963 as Project MAC (for Multiple Access Computer and Machine Aided Cognition), the Laboratory developed the Compatible Time Sharing System (CTSS), one of the first time shared systems in the world, and Multics -- an improved time shared system that introduced several new concepts. These two major developments stimulated research activities in the application of on-line computing to such diverse disciplines as engineering, architecture, mathematics, biology, medicine, library science and management. Since that time, the Laboratory's pursuits expanded, leading to pioneering research in Expert Systems, Computer Networks and Public Cryptography. Today, the Laboratory's research spans a broad front of activities, grouped in four major areas:

The first such area entitled *Knowledge Based Systems*, involves making programs more intelligent by capturing, representing, and using knowledge which is specific to the problem domain. Examples are the use of expert medical knowledge for assistance in diagnosis carried out by the Clinical Decision Making Group; and the use of solid-state circuit design knowledge for an expert VLSI (very large scale integration) design system by the VLSI Design Project.

Research in the second and largest area, entitled *Machines, Languages, and Systems*, strives to discover and understand computing systems at both the hardware and software levels that open new application areas and/or effect sizable improvements in their ease of utilization and cost effectiveness. New research in this area includes the architecture of very large multiprocessor machines (which tackle a single task, e.g., speech understanding or weather analysis) by the Computation Structures, Real Time Systems, Information Mechanics, and Parallel Programming Research Groups. Continuing research includes the analysis and synthesis of languages and operating systems for use in large geographically distributed systems by the Distributed Systems and Programming Methodology Groups. Finally, a key application involving the matching of news and other community information to individual needs, is pursued by the Programming Systems Research Group.

The Laboratory's third principal area of research, entitled *Theory*, involves exploration and development of theoretical foundations in computer science. For example, the Theory of Computation Group strives to understand ultimate limits in space and time associated with various classes of algorithms, the semantics of programming languages from both analytical and synthetic viewpoints; the logic of programs; the utility of

randomness in computation; concurrent computation and the links between mathematics and the privacy/authentication of computer-to-computer messages. Other examples of theoretical work involve the study of distributed systems by the Theory of Distributed Systems Research Group, and the development of effective algorithms for VLSI design.

The fourth area of research, entitled *Computers and People*, is concerned with the interrelationships between people and machines -- for example, the societal impact of computers carried out by the Societal Implications Research Group.

Some of the research highlights were as follows:

The Multiprocessor Emulation Facility (Prof. Arvind) was rendered operational and was used for its intended purpose of emulating our Laboratory's Tagged-Token Dataflow architecture. This facility, currently consisting of 38 Lisp Machines which are interconnected by Laboratory designed high speed switches, was designed to be a testbed for experimental study of various candidate multiprocessor architectures. The results of these first emulations and related studies have led us to revise our designs for a tagged dataflow machine, which we hope to build within the next three years. A comprehensive software system has also been completed that translates functional programs from the language Id to representations that can be emulated on the above facility, simulated on an IBM 4381 computer, or executed on a dataflow machine.

In addition, we have also launched the design of two additional multiprocessor projects: Project L (Prof. Ward) and CAM-7 (Dr. Toffoli). L is a new model of computation characterized by (1) a large collection of finite state machines and state representations with the property that programs written in object-oriented languages (with concurrency) can be efficiently compiled into L structures; and (2) L structures can be efficiently executed on a proposed hardware architecture associated with the L project.

The CAM-7 architecture is a cellular automata machine that further extends our previous architectures in this area. This machine will be able to update, in one screen refresh interval, half-a-billion digital cells which are configured either as a cube or as a plane. The state of each such cell is updated on the basis of the states of its physically adjacent cells, as dictated by a set of rules that apply uniformly to all cells. We use these cellular automata structures for a variety of purposes including the development of simple computational models that explain relatively complex physical phenomena.

In January 1986, we formally launched the LCS Common System research effort. This ambitious project aims to facilitate the composition of programs across different computational environments. For example, a program written in a Lisp machine environment should be able to use a subprogram written in C under a Unix

environment. We expect to complete the first definition of this system by year's end, at which point we will begin gradual implementation of the Common System on the Laboratory's 100 Lisp Machines and VAX-2 workstations.

An additional research activity launched in 1985-86 involves learning. This "Artificial Awareness" program (Prof. Rivest), as it is called, strives to develop theories and machines that can learn from their environments (and not from their programmers) certain elementary notions. We believe that technological progress in architectures and VLSI calls for a re-examination of learning theories and learning machine architectures that may perhaps be implemented by "neural" types of machines.

Each year, the Laboratory holds its successful Distinguished Lecturer Series. During the 1985-86 academic year presentations were given by Princeton University Professor Robert E. Tarjan; Executive Director of Research at AT&T Bell Laboratories' Communications Sciences Division, Robert W. Lucky; Cornell University Professor John E. Hopcroft; Tufts University Professor of Philosophy Daniel C. Dennett; Chairman of the Board of Microsoft, William H. Gates; and University of Kent at Canterbury Professor David A. Turner.

The Laboratory had several personnel changes over the past year including the addition of Mathematics Department Professors Baruch Awerbuch and David Shmoys, and Dr. Leanord Heath as members of the Theory of Computation Group; the arrival of Drs. Sharon Marshall, Stephen Garland, and Toby Bloom as research associates in the Clinical Decision Making, Systematic Program Development, and Distributed Computer Systems Groups, respectively; the departures of Professor Christopher Terman to Symbolics, Inc., Dr. Andrea diSessa to Berkeley, Research Associate Dr. William Ackerman to Apollo Computer, Research Associate Sylvia Weir to the MIT Arts, Media & Technology Center, and Adjunct Professor Maurice V. Wilkes, who returned to his home in the United Kingdom. Other changes included the appointments of Dr. Tommaso Toffoli to head of the Information Mechanics Group, Dr. Andy Boughton as Research Associate in the LCS Multiprocessor Emulation Facility, and Professor Robert H. Halstead as head of the newly formed Parallel Programming Research Group. Additionally, the Functional Languages and Architectures, and Computation Structures Groups have merged, retaining the name of the latter.

Our Laboratory consisted of 331 members -- 44 faculty, and academic research staff, 30 visitors and visiting faculty, 62 professional and support staff, 105 graduate and 90 undergraduate students -- organized into 15 research groups. Laboratory research during 1985-86 was funded by 12 governmental and industrial organizations, of which the Defense Advanced Research Projects Agency of the Department of Defense provided over half of the total research funds. In addition, the Laboratory employed 23

undergraduates through the "Hacker Heaven" project which strives to identify promising potential researchers in Computer Science.

Our technical results in 1985-86 were disseminated through publications in technical literature, through Technical Reports (numbers 356 through 366) and through Technical Memoranda (numbers 291 through 309).

CLINICAL DECISION MAKING

Academic Staff

P. Szolovits, Group Leader

R. Patil

Collaborating Investigators

M. Criscitiello, M.D., Tufts-New England Medical Center Hospital
M. Eckman, M.D., Tufts-New England Medical Center Hospital
R. Friedman, M.D., University Hospital, Boston University
W. Hardy, Ph.D., University Hospital, Boston University
J. Kassirer, M.D., Tufts-New England Medical Center Hospital
M. Klein, M.D., University Hospital, Boston University
S. Kurzrok, M.D., Tufts-New England Medical Center Hospital
J. Lau, M.D., Boston VA Medical Center
A. Levey, M.D., Tufts-New England Medical Center Hospital
A. Moskowitz, M.D., Tufts-New England Medical Center Hospital
S. Naimi, M.D., Tufts-New England Medical Center Hospital
S. Pauker, M.D., Tufts-New England Medical Center
W. Schwartz, M.D., Tufts-New England School of Medicine
F. Sonnenberg, M.D., Tufts-New England Medical Center Hospital
L. Widman, M.D., Case Western Reserve University Medical School

Research Staff

G. Burke
R. Jayes
W. Long
S. Marshall

Graduate Students

D. Fogg
H. Goldberger
R. Granville
D. Hirsch
P. Koton
R. Kunstaetter
S. Novick
T. Russ
E. Sachs
D. Smit
M. Wellman
A. Yeh

Undergraduate Students

T. Chow
I. Haemowitz
C. Kim
K-Y. Lai
T-Y. Leung
D. Pachura

Support Staff

R. Hegg

1. INTRODUCTION

Members of the Clinical Decision Making Group and their colleagues at Tufts/New England Medical Center have been involved in a variety of research activities under the auspices of our two main projects entitled *An Artificial intelligence Approach to Clinical Decision Making* and *A Program for the Management of Heart Failure.* In the former project, led by Prof. P. Szolovits and described in the following section, we are trying to build a large coherent medical diagnosis and therapy system. The final section describes how in the Heart Failure Project (sometimes referred to as CHF - Congestive Heart Failure project) Dr. W. Long and his colleagues are developing a methodology and tools for assisting the physician in reasoning about the diagnosis and management of heart failure.

2. AN ARTIFICIAL INTELLIGENCE APPROACH TO CLINICAL DECISION MAKING

The group is pursuing a multi-pronged attack on the technical difficulties of creating intelligent medical reasoning programs within the computer. The fundamental assumption of our current efforts, and thus the intellectual basis for our current work, is that it is not sufficient to solve single independent technical problems. Instead, we are attempting to build a large, coherent computer system that embeds a set of complementary techniques and bodies of knowledge which will enable us and our colleagues to build functioning medical diagnosis and therapy planning and monitoring programs for a significant range of medical domains.

The principal research components of the above plan are:

1) to develop a knowledge representation language and a set of conceptual primitives adequate to express the kinds of medical knowledge we need to encode in the rest of the project;

2) to work out the particular methods of using such a language to encode the particular knowledge of a small set of medically-interesting but narrow specialties (initially, fluid/electrolyte balance, and coronary artery disease);

3) to implement and re-implement a variety of reasoning methods that our own group and others have developed during the past decade for medical problem solving, but now in such a way that they can all take advantage of the uniform knowledge representation and share knowledge and techniques;

4) to build a collection of exemplary cases and case analyses for use in building and testing the reasoning programs;

5) to develop new means of reasoning with incomplete, or qualitative, descriptions of physiologic processes, probabilities, utility structures, etc.;

6) to apply artificial intelligence modeling methods to the clinical practice of decision analysis; and

7) to integrate methods of decision analysis into AI-based reasoning programs.

To address point 1, we have developed a good working relationship with the community of computer scientists developing and using the NIKL knowledge representation language (our initial choice of a base on which to build), acquired working versions of that language, and conducted a number of small to medium-sized experiments to determine how the facilities of that language support our needs. Prof. R. Patil has implemented in NIKL several alternative versions of part of the knowledge represented in the ABEL program, to explore how best to represent notions such as causal relations and quantitative dependencies. Because many of these representational issues have not previously been addressed by developers or users of NIKL, we have found that considerable additional design will be needed to support these capabilities elegantly and efficiently.

In the short term, both to learn more about how to best exploit NIKL and to address point 2 in our plan, Prof. Patil and Dr. Long have created an intermediate representation language that is mapped into NIKL but allows us to change how that mapping is performed. Prof. Patil and Mr. I. Haemowitz have used this language to implement almost all the non-procedural medical knowledge of ABEL, and Ms. T. Chow has begun to build a text and graphics browsing tool to permit the exploration of the knowledge base expressed in this form. Dr. F. Sonnenberg has taken the lead in determining what are the principal issues in the representation of anatomical concepts, Prof. Patil, Drs. J. Kassirer, R. Jayes and O. Senyk have encoded an initial version of the detailed knowledge of potassium metabolism, and Dr. R. Jayes has encoded considerable detailed knowledge of the apparently relevant literature of coronary artery disease and the drugs that are used in its treatment. Although we have not yet been able to settle on the appropriate conceptual primitives, the intermediate language approach has permitted us to make progress on the collection of knowledge.

We have done some exploration of how to import reasoning methods (point 3) such as those from ABEL, EVOKER, the CHF (Congestive Heart Failure) program, and Internist, and Mr. C. Kim implemented a Reconsider-like diagnosis program based on the AMA's CMIT database. Because we have concentrated much of our efforts on how appropriately to represent knowledge, we have used this exploration more to guide that representation effort rather than to build significant new programs. Based on some of these methods, Dr. R. Kunstaetter, working with Dr. G. O. Barnett at the Massachusetts General Hospital and with Prof. Szolovits, completed a medical teaching program for respiratory physiology that was tested by students in the Harvard Medical School New Pathway curriculum.

Ms. P. Koton has begun to develop a program to index cases (point 4) previously encountered by our programs and to permit the use of case-based analogic reasoning. Working with Dr. S. Pauker, she has collected approximately 20 cases to date, all in the cardiac area. In addition, we have begun to catalog all the decision analysis consults done by the Tufts/NEMC group (see below).

One area where significant new AI results are clearly necessary is in reasoning about partially-described systems (point 5). Mr. E. Sacks has continued his development of the Qualitative Mathematical Reasoning Program, extending it to begin to deal with non-linear problems by making suitable linear approximations over limited regions. Mr. M. Wellman has continued to apply similar methods to the analysis of the structure of dependencies in probabilistic networks and among related health outcome descriptors. Dr. L. Widman, collaborating with us remotely from Case, has developed some methods of semi-quantitative analysis, as a more traditional, simulation-oriented alternative to qualitative techniques. The fundamental approach here is to turn qualitative judgments into quantitative estimates, use variants of simulation algorithms to predict the future state of the world, and then reinterpret that state in term of the qualitative states of interest.

We have concentrated heavily on developing new technical means of improving the clinical use of decision analytic methods (point 6) for making difficult choices based on a careful analysis of risks and benefits to the individual patient, employing the patient's own preferences as a basis for determining formal utilities. One major aspect of this work has been to develop a decision-tree critiquer (DTC) that comprehends typical errors made in the application of these methods and suggests improvements in the analysis. Mr. Wellman and Dr. Eckman have built an initial version of such a program, and they have cooperated with Dr. S. Marshall in beginning the construction of a catalog of analyses and errors, from which the expertise of the DTC will be derived. Dr. Sonnenberg has also developed new methods of automatically revising context-dependent variables in decision trees.

Though we have had extensive discussions of the appropriate means of including probabilistic reasoning in AI programs, we have not yet made much progress on point 7. We have discussed these issues with leading visiting researchers, and we have begun to try to apply the qualitative reasoning methods we are developing to these problems.

Overall, we have made a constructive start on each of the long-term parts of our research plan, and have made substantial progress on knowledge representation (1), knowledge encoding (2), qualitative reasoning (5) and applying AI to decision analysis (6).

A report on our research results and plans in each of the main research areas follows:

2.1. Knowledge Representation

The central goal of our long-term research effort is to develop a modular, integrated knowledge representation formalism in which all the medical and other real-world knowledge relevant to medical decision-making can be encoded and manipulated. As proposed, we have initially adopted the knowledge representation language NIKL (New Implementation of KL-ONE) as the basis for our work. NIKL, being an experimental general-purpose knowledge-representation tool, does not, of course, solve the problems of how to represent medical knowledge in the computer. It does, however, provide a set of epistemological primitives for forming descriptions, inferring when one description necessarily subsumes another, and displaying the descriptions entered and their interrelationships in both textual and graphical ways. Our efforts with NIKL have centered on three tasks: (1) get a current version of it working in our environment, and establish good working relationships with other major NIKL users and with developers of the system at USC/ISI and at BBN; (2) design the appropriate mappings between medical concepts we are interested in representing and the best ways of achieving those representations in NIKL; and (3) where needed, design and implement extensions to NIKL, in collaboration with other users, to reflect newly-recognized needs identified by our use of the language.

During the past year we have successfully imported the NIKL knowledge representation software and developed in-house expertise in its use for medical knowledge representation. Under the leadership of Prof. Patil, we have translated much of the ABEL (Acid-Base and Electrolyte Project) knowledge base into the NIKL representation and are currently in the process of re-implementing the ABEL program on the Lisp Machine, using the NIKL knowledge base. We have also developed a prototype intermediate knowledge acquisition language for acquisition of medical knowledge which is then translated into input language of the target knowledge representation system (currently NIKL). The prototype language is implemented in a table-driven fashion, allowing us to incrementally extend the input language as well as to reorganize the representation of this knowledge in the target knowledge representation system without the need for manually recoding the existing knowledge base. We have used this intermediate knowledge acquisition language to develop a model of the circulatory system adequate for both the Acid-Base and Electrolyte Project and the Congestive Heart Failure project. We are currently in the process of developing a hierarchically organized knowledge base of medical terms and concepts to be shared between these two projects. In addition, we have built graphical tools to permit the interactive investigation of the content of the knowledge expressed in the acquisition language, to permit physicians without extensive computer science training to examine and modify that knowledge.

Mr. S. Balzac has been working under the supervision of Prof. Patil in exploring and implementing extensions to NIKL system motivated by the needs of unified knowledge base. We have found the representation of the PART/WHOLE relation in anatomical models to be particularly problematic in the existing NIKL implementation. The problem arises because the existing NIKL and its classification algorithm is based on the assumption that all terms in the knowledge-base can be organized into a single *type* hierarchy based on the *subsumption* relation. We have found this restriction constraining in our domain. To overcome this difficulty, we have been engaged in developing an extension to NIKL which will allow a number of different type hierarchies and will allow a concept to be defined using more than one type hierarchy. We expect that the first version of this program will be completed early in the fall and will be used to implement a part of the knowledge-base that is currently expressed in the intermediate representation language but which has no adequate translation into NIKL. The adequacy of this proposal will be evaluated, and, if successful, we will make the changes available to the broader NIKL community and adopt the new version as our knowledge representation language.

We have also convened a national workshop on the use of NIKL, held at MIT in July 1986, bringing together 20 of the developers and principal users of this language, to discuss our respective experiences with it, needed changes that have been identified through actual use of the language, and experiences from other knowledge representation projects that may be relevant to future NIKL development. Some of the issues raised at this meeting, which will be the focus of additional work in the coming year, include:

- the relationship between terminological reasoning and other forms of inference, such as individuals, general assertions, defaults and assumptions, etc.;
- the possibility of supporting not only definitional (necessary and sufficient) conditions in the terminologic language but also weaker forms of description, such as (perhaps) alternate sufficiency conditions no one of which may be necessary;
- more complete support for inverse roles, including their handling by the classifier;
- integration of the systems's assertional component with a more conventional data base system, to support the handling of large bodies of simple factual knowledge;
- extension of the semantics of the language to deal with concepts such as sets and sequences, part/whole relations, etc.; and
- extension of the language to handle meta-level descriptions, including self-description.

Our own interest in these issues is focused by the needs we have identified for support of our medical representation work.

One of our high-priority tasks is to explore extensions to NIKL and development of better tools for maintaining and extending the knowledge base in NIKL. Based on experience gained in modeling medical knowledge in NIKL, we have identified a number of shortcomings in the NIKL formalism. For example, in the anatomical domain we have found that the mechanisms for describing PART/WHOLE relationships in NIKL is inadequate (see above). In describing knowledge of diseases, we have noted that facts such as "MI is more common among men than women" cannot be expressed in NIKL, which has no present capability for making uncertain statements. As our knowledge base has grown over the last few months to include over 1500 concepts, we have discovered that the utilities provided in NIKL for interactive graphical display and browsing are inadequate for dealing with large knowledge bases. We have proposed a number of enhancements to NIKL to the NIKL development groups at ISI and BBN. Over the next year we expect to continue to cooperate with these groups to extend the NIKL language to overcome fundamental difficulties, and we expect ourselves to develop those facilities (such as displaying/browsing through the causal networks and patient-specific models) which are specific to the medical domain.

We plan to develop and extend the intermediate knowledge acquisition language and other knowledge acquisition tools. Over the years, our group has been involved in a number of projects using a variety of knowledge representation languages. As a result we have been forced to develop knowledge bases for each project *de novo*, even when part of the knowledge base needed for a new project was already present in existing knowledge bases. To insulate us from this problem in the future, we have developed a prototype intermediate language for describing medical knowledge. The prototype language is implemented in a table-driven fashion allowing us to incrementally extend the input language as well as to reorganize the representation of this knowledge in the target knowledge representation system without the need for manually recoding the existing knowledge base. As the work towards acquisition of medical knowledge progresses and new constructs are identified, we will continue to extend the intermediate language to include them. We also expect to develop an interface to translate the internal structures of the intermediate language into simple English and ultimately to provide a restricted subset of English as an input language which will greatly enhance the ability of clinical experts to participate directly in the acquisition of medical knowledge.

In addition to the above efforts in knowledge acquisition, Mr. R. Granville has been exploring a methodology called "stratification," a form of structured layering of knowledge for building expert systems. The goal of this study, similar to the earlier

goals of the "structured programming" school of traditional programmers, is to provide principles that help assure modularity of large knowledge bases. His work is being conducted in the context of a system that accepts pseudo-English descriptions of pieces of knowledge and constructs both a linguistically-oriented case-frame representation that is then used for generating explanations and a Lisp function that can be run to apply the bit of expertise that was expressed.

2.2. Development of a Uniform Knowledge Base

Our work in developing new knowledge representation formalisms and our work in actually encoding the relevant medical knowledge of the disease areas we are interested in exploring must, innately, go hand-in-hand. We typically come to appreciate the strengths and weaknesses of any particular representation method only as we challenge it with actual knowledge to be encoded. Therefore, both as an experimental vehicle to support our investigation of representation methods and as the start of our long-term efforts to compile a usable, formally-represented knowledge base, we have begun to encode medical knowledge in the fields of coronary artery disease and acid-base and electrolyte disorders.

Dr. R. Jayes has been actively preparing summaries of the information needed to accurate diagnosis and treatment in the coronary artery disease domains, including a taxonomy of necessary anatomical and clinical concepts, as well as ancillary knowledge such as the drugs ordinarily used in treatment, their characteristics, risks and benefits. He has also reviewed the decision analysis literature in this field, to assure that principles accepted there are incorporated in the knowledge base. An example of part of this knowledge, encoded in the intermediate representation language we are using, is shown in Figure 2-1.

Dr. Jayes, Prof. Patil and Mr. Haimowitz have also been creating a database for modeling physiological events, diagnosis, and therapy in the acid-base and electrolyte domain. Part of this knowledge has come from a review of the literature and part from a translation of the knowledge base of the ABEL program into our new representation formalism. The anatomic structure and connectivity of parts of the kidney are represented in Figure 2-2. The causal structure of a portion of the ABEL domain is shown at three levels of detail in Figure 2-3.

In each of the domains we have addressed, we have encountered some systematic problems of representation that demand general solutions for representing medical concepts. Unlike the totally general issues like part/whole relations mentioned above, these problems are definitely medical in nature; thus we anticipate that solutions to these will be relevant to a broad range of projects in our own work, but not likely to be significant to more distant representation efforts.

```
(c circulatory-circuit-through-lungs (s circulatory-circuit))
(c artery
   (c pulmonary-trunk (s artery)
      (part-of circulatory-circuit-through-lungs)
      (input right-ventricle)
      (output main-pulmonary-arteries))
   (c main-pulmonary-artery (s artery)
      (part-of circulatory-circuit-through-lungs)
      (input pulmonary-trunk)
      (output large-branches-of-pulmonary-artery))
   (c large-branches-of-pulmonary-artery (s artery)
      (part-of circulatory-circuit-through-lungs)
      (input main-pulmonary-artery)
      (output pulmonary-arterioles))
   (c pulmonary-arterioles (s artery)
      (part-of circulatory-circuit-through-lungs)
      (input large-branches-of-pulmonary-artery)
      (output pulmonary-capillaries)))
(c capillaries
   (c pulmonary-capillaries (s capillaries)
      (part-of circulatory-circuit-through-lungs)
      (input pulmonary-arterioles)
      (output pulmonary-venules)))
(c vein
   (c pulmonary-venules (s vein)
      (part-of circulatory-circuit-through-lungs)
      (input pulmonary-capillaries)
      (output large-branches-of-pulmonary-veins))
   (c large-branches-of-pulmonary-veins (s vein)
      (part-of circulatory-circuit-through-lungs)
      (input pulmonary-venules)
      (output pulmonary-veins))
   (c pulmonary-veins (s vein)
      (part-of circulatory-circuit-through-lungs)
      (input large-branches-of-pulmonary-veins)
      (output left-atrium)))
```

Figure 2-1: Portion of the Circulatory Model Concerning the Lungs.

One such problem area we have been exploring is the representation of anatomy. The major conclusion of this work so far is that at least the following aspects of anatomical knowledge need to be represented:

- Morphology -- the size, shape, color and texture of objects.
- Topology -- the relationships of objects in space.
- Internal Structure -- the subcomponents or composition of an anatomical entity.

```
(c kidney (s organ)
   (c left-kidney)
   (c right-kidney))
;;; these are organ parts for kidney
(c organ-part
   (c renal-interstitium (s organ-part)
      (part-of kidney))
   (c renal-cortex (s organ-part)
      (part-of renal-interstitium))
   (c renal-medula (s organ-part)
      (part-of renal-interstitium))
   (c renal-blood-vessel (s blood-vessel)
      (c renal-arterie (s artery)
         (input systemic-artery)
         (output renal-vein))
      (c renal-vein (s vein)
         (input renal-arterie)
         (output systemic-vein)))
   (c nephron (s organ-part)
      (part-of kidney))
   (c tubule (s anatomical-conduit organ-part)
      (part-of nephron)
      (input glomerule)
      (output collecting-duct))
   (c proximal-tubule (s anatomical-conduit organ-part)
      (part-of tubule)
      (input glomerule)
      (output loop-of-henle))
   (c loop-of-henle (s anatomical-conduit organ-part)
      (part-of tubule)
      (input loop-of-henle)
      (output distal-tubule))
   (c distal-tubule (s anatomical-conduit organ-part)
      (part-of tubule)
      (input loop-of-henle)
      (output collecting-duct))
   (c glomerule (s organ-part)
      (part-of nephron))
   (c collecting-duct (s anatomical-conduit organ-part)
      (part-of kidney)
      (input tubule)))
```

Figure 2-2: Part of the Anatomical Model for the Kidney.

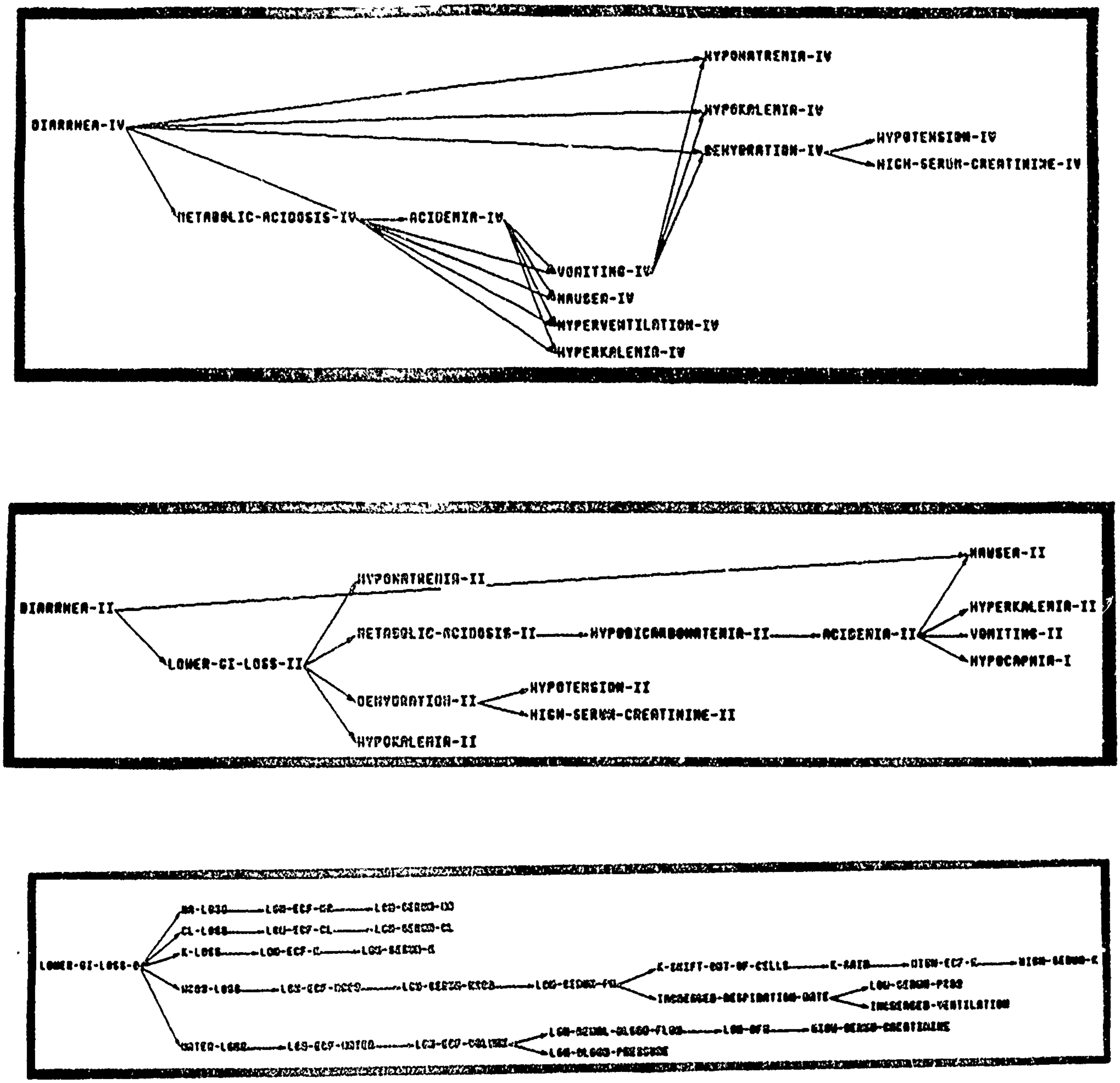

Figure 2-3: Causal Chains for Acid-Base Disturbances.

The above three diagrams show the causal consequences of diarrhea at three successively more detailed levels in the ABEL database. These diagrams were produced from the NIKL encoding of this data, using causal structure-tracing utilities developed as part of the project.

- Taxonomy -- upward references specifying of which anatomical entity some other entity is a part.
- Commonsense Knowledge -- this aspect includes "generic prototypes" of which some given anatomical entity is an instance.

The taxonomy which organizes medical knowledge is a key to the usefulness of this knowledge in medical diagnosis. The taxonomy spans a number of levels from the most macroscopic, namely, the entire body, through regions, organ systems, tissues, cells, organelles and ultimately molecules. Each medical reasoning task will address primarily one of these levels, and the most clinically relevant knowledge available for a particular task will also apply primarily to one level.

The taxonomic organization of anatomic structures lends itself to encoding in a structured inheritance network such as NIKL. We have experimented with encoding anatomical knowledge using NIKL. Future work planned includes exploration and summarizing of the limitations of the use of NIKL to encode anatomical knowledge so as to suggest representation schemes which will avoid these limitations.

A second generic medical representation issue that has arisen is the appropriate treatment of multiple taxonomies. It has become clear from preliminary work done by our research group and others that there is no single hierarchical scheme which is capable of containing all of medical knowledge or of providing all of the necessary inferential links from one piece of knowledge to the others. Work has focused on the nature of these multiple hierarchies to define how many different types are possible, how many different types are clinically relevant and how an inference scheme should choose among them when formulating hypotheses. This work has so far revealed that among the major hierarchies are included anatomical, etiologic, physiologic, and temporal hierarchies. Each of these organizes available knowledge in different ways. Most medical concepts fit into all of these hierarchies. Future work will concentrate on constructing a scheme to encode medical knowledge in a multiple-based hierarchy to explore the inferences which can be made from them.

In the coming year we expect to complete the representation of the circulation model and the functional anatomy model as the first steps toward a unified medical knowledge base of terms and concepts. A number of issues in this area still remain to be resolved. For example, we need to develop representations for describing arterial shunts where the blood can flow in either direction. This is particularly difficult using symbol-level structures in NIKL because they implicitly assume unidirectional connections. Similarly, there are a number of anatomical structures which are patient-dependent (e.g., left vs. right dominated coronary blood supply). Currently we do not have the capability of describing these alternate models or selecting one of them for reasoning in a specific case. In addition to overcoming these difficulties we will also be working on developing

models for homeostatic processes relevant to ABEL project: the HCO_3-$PaCO_2$-pH regulation, salt and water regulation, and potassium metabolism.

2.3. Importing Components of Existing Reasoning Systems

We plan to complete the new version of the ABEL program and enhance its medical knowledge base beyond the current system. This will allow us to explore in more detail the limitations of the ABEL program and provide us with ideas for redesign of Enhanced ABEL system. By late next year we expect to begin the conceptual design of the new ABEL system. There are a number of features of the new system, which have, already emerged from past experience. (1) The new system will not maintain a rigid stratification between the levels of detail throughout the knowledge base. This will allow us to expand different parts of the PSM tc different levels of detail; opening those areas where interaction among diseases must be resolved to great detail while maintaining at relatively shallow level, other areas, where the diagnostic reasoning is relatively straight forward. (2) The process of proposing new hypotheses in ABEL is achieved through causal propagation of unaccounted for findings. This process, although satisfactory, is computationally quite expensive. For example, during diagnosis, each cycle of hypothesis generation takes approximately five times more computation than revision in the multi-level PSM to reflect the result of the diagnostic cycle. In the newer implementation we expect to combine the heuristic hypothesis generation techniques developed in PIP, INTERNIST/CADUCEUS projects to reduce this time considerably thus making the overall program significantly more efficient.

In addition, we plan to bring into the system we are building a number of other methods we and our colleagues have developed in other projects. In the case-based reasoning efforts, we plan to use the methods developed independently by Dr. Long in the CHF project. We also plan to build an EVOKER-like utility to provide a general PIP or Internist-I style hypothesis evocation tool.

2.4. Representation and Organization of Case-Specific Knowledge

We have identified two top-level issues in representing knowledge about particular patient cases to be dealt with by our system. One is simply the ability to represent what is actually known about a patient in the same language in which general medical facts are expressed, so that it becomes possible to match descriptions of the individual against prototypical cases. The other is the more difficult problem of organizing large numbers of such individual case descriptions to permit intelligent retrieval of cases based on features of interest and (ultimately) the possible use of analogic, case-based reasoning to allow the system to recognize that an analysis of a previous case or set of cases is highly relevant to the current case under consideration.

The only problem we have identified in the first of these issues is that the NIKL language does not have specific support at the moment for the representation of individuals. Thus, although any concept can be marked as an individual, the intended semantics of this marking is more that "this concept can have only a single instance" (e.g., "the president of IBM Corp.") rather than that the concept directly stands for a particular entity in the real world (e.g., Joe Smith). A similar problem arises in the characterization of roles. Because NIKL is intended to describe generic concepts, one might restrict the *age* role of a *person* to be a *number*, or perhaps *number less than 100*, and it is difficult to store the actual value, rather than a descriptor, in the slot. This problem will be resolved either in some appropriate general way by developing a comprehensive facility for representing individuals in NIKL or by adopting some *ad hoc* convention. In either case, we expect it not to be a difficult issue once the more general medical knowledge representation issues are resolved.

The more difficult and interesting problem is that of organizing a large number of cases in such a way that automatic indexing structures help to retrieve just the most relevant cases when case-based analogic knowledge appears to be called for. P. Koton has developed her doctoral research proposal around a new approach to this problem. She plans to use an indexing scheme similar to that developed by Dr. J. Kolodner of Georgia Tech in her work on memory structure. This approach involves indexing cases by their salient features, and constantly reformulating a taxonomic system in which cases that are most similar along one or another dimension are grouped together and eventually define prototypes. The most important of Koton's adaptation of this system is to introduce the causal explanations developed in past cases as one of the most important features for indexing, in contrast to the more typical methods that rely simply on similarity of surface appearance to determine closeness. In addition, we plan to avoid some of the problems that have troubled this approach by building a much more comprehensive index, permitting the selection of salience to be made at retrieval, rather than indexing time.

2.5. Qualitative Methods of Reasoning

Mr. E. Sacks has worked on several aspects of his doctoral thesis project, qualitative analysis of systems described in engineering terms. First, he has assessed the available mathematical tools, the theories of linear and nonlinear differential equations. In principle, the linear theory provides a complete analysis of all linear differential equations, although in practice, computational difficulties may arise. The nonlinear theory is very sketchy and difficult to apply. Next, he investigated how several engineers analyze nonlinear systems. They appear almost always to reduce nonlinear systems to piecewise-linear ones. Therefore, the program he is developing will take the

same approach. So far, part of the linear systems analyzer and several utilities have been written, including an algebraic constraint reasoner, an inequality solver and a graph sketcher.

2.6. Integration of AI and Decision Analytic Methods

During the past year we have begun a major attack on the problems of how to utilize decision analysis in clinical medicine by bringing AI techniques to bear in the construction, evaluation and interpretation of decision analyses. We have also begun the exploration of how best to use the insights from decision analysis in those parts of AI decision-making that rely on quantitative judgments of likelihood and value. The Lab has had a number of visitors (J. Pearl, P. Cheeseman), with whom we have pursued the applicability of "standard" but extended Bayesian models, and we have pursued the application of new qualitative mathematical reasoning techniques to problems of incomplete and inexact knowledge of likelihoods and preferences.

Mr. M. Wellman has been one of those working on techniques for medical decision making that integrate artificial intelligence and decision theory methodologies. His accomplishments over the past year include the design of a knowledge representation of health outcome descriptors to be used for automatically formulating decision models from a large medical knowledge base, implementation of decision tree software for Symbolics Lisp Machines that will evolve into a program that structurally analyzes and critiques users' decision models (see below), and substantial improvements to an algorithm for automatically selecting a multi-attribute utility decomposition given independence conditions asserted in a knowledge base. In addition, he has developed a qualitative algebra for probabilistic networks that provides for "structural" reasoning about likelihoods, and therefore appears promising for planning under uncertainty.

2.7. Decision Tree Critiquer

One of the first concrete applications of AI technology to decision analysis is the construction of a Decision Tree Critiquing (DTC) program. The construction of decision analytic models is a non-trivial skill that requires much practice and experience. As more medical personnel become interested in building decision tree models of their own, there will develop an increasing need for systems which can interactively assist in the construction of these medical decision trees. Toward this end, we have been developing a new decision tree software environment that will attempt to critique user-developed decision models both while they are being constructed and after their completion.

Several types of knowledge are required for the program to offer sound and effective

advice to the decision tree builder. It is our hypothesis that much about a decision tree can be appreciated from a purely structural plane, independent of the medical domain knowledge. Much is gained, as well, from an understanding of the kinds of mistakes which are commonly committed. The Clinical Decision Making service at Tufts New England Medical Center has provided a rich source of experience, both in terms of the principles of good model construction and in the observation of experienced tree builders criticizing the many models presented by students rotating through the service.

Examples of purely structural considerations in tree construction are meaningful symmetry among strategy branches, the exclusion of strategies consisting of tests without risk, and the inclusion of information dependencies within the representation, either via the explicit binding of arguments to parameters of a node or a parallel representation graph of causal linkages. Probability issues include the automatic calculation of Bayesian revisions based upon test results, prevalence data, and sensitivity/specificity information, the calculation of the probability of various test results at chance nodes, and the handling of conditional probabilities in serial chance nodes. With respect to issues requiring medical domain knowledge, the "critiquer" needs to understand utility structures and the situations in which different types of metrics are applicable, causal dependencies and the resultant linkages that must exist between like parameters on multiple strategy branches, and information on disease prevalence and test characteristics.

Work to date includes the design of a critiquing architecture, implementation of the more standard decision tree software on the Lisp Machine, and documentation of many of the issues in medical decision tree construction that we expect our critiquer to handle. Over the next year we plan to define and implement a language for specifying critiquing rules that will use the system's built-in structural analysis routines to recognize and respond to decision tree anomalies. The system will be exercised substantially in both developmental testing and routine use by the consult service. This use will provide for systematic collection of examples of decision tree bugs for further refinement of the critiquing knowledge.

Presently, the higher level modeling skills form the more "artistic and creative" aspects of decision analysis. Work on the decision tree critiquer will help formalize this process, removing some of the mystery while promoting the technique's greater dissemination and availability.

Dr. Marshall, Dr. Eckman and Mr. Wellman have begun to catalog errors made by beginning medical decision analysts. We are interested in errors in structuring trees, reasoning about uncertainty and calculating with probabilities, describing outcomes, and assigning utilities. Our motivation for developing such a catalog is two-fold. First, we

expect that the catalog will be a rich set of material to use in teaching, and improving, medical decision analysis. Second, we hope to generalize the material in the catalog to further our understanding about how people acquire problem solving skills.

The material for the catalog is being collected at New England Medical Center (NEMC). Three new fellows arrived in the Division of Clinical Decision Making on I July 1986 and they have begun to participate in the patient consultation service. Many of the consults present problems that are amenable to decision analysis. Active consults are presented formally semi-weekly and are discussed informally as needed. Since we participate in the formal discussions, and many of the informal ones, as well, we have ample opportunity to observe the Fellows' learning process.

The catalog will contain a short precis of each case (written by the medical fellow to whom the consult has been assigned), together with not only a recommendation and supporting analysis, but also documentation of preliminary analyses, usually with several iterations, our observations of errors and comments. The development of the catalog is facilitated by the DTC program that is being developed in our group. The program allows us to structure and modify decision trees with a minimum of effort. DTC also enables us to analyze trees and arrive at an optimal strategy based on the maximization of expected utility.

In order to capture the structural knowledge necessary for the analysis of decision trees, we developed a notation which creates a hierarchical ordering of node types: decision, chance, and terminal. Nodes are specialized through A-KIND-OF links. Treat-decisions, test-decisions, and strategy-decisions are all kinds of decisions. Test-results, treatment-efficacies, physiologic-states, and complication-events are all kinds of chance events. Each node type has certain associated tasks. Each physiologic-state node, for instance, determines the influences of its different states upon downstream events in the tree. It must also determine what the effect of "upstream" tests and treatments may be on the distribution of states across that chance node. As a further example, complication-event nodes must look "upstream" to find their associated test or treatment (from which the complication arose), and also determine if any of the "distinguished" complications (globally defined complications) such as death are present. In a similar manner, each node type has tasks it must fulfill in being called.

In our work with applied decision analysis we often encounter the problem of revision of context-dependent variables. An example of this is changing the probability of disease based on the results of a diagnostic test for nodes of the tree "downstream" from the test result. Faulty probability revisions are a very common cause of erroneous decision analysis results and are very difficult to find and correct. Dr. Sonnenberg has derived a method for checking to see that these probability expressions are correct, and

an algorithm for deriving them. The algorithm requires delineation of an event space for the clinical situation and specification of the dependencies among various events. The methodology discussed here has been applied successfully to a real-life decision analysis involving the generic policy issue of evaluating pulmonary disease in patients suspected of having the Acquired Immune Deficiency Syndrome (AIDS). The event space for this analysis is so complicated that derivation of the necessary probability expressions could not be done without an algorithmic approach. Work on this real clinical problem has enabled development of the techniques for generation of probability expressions and verification of the result.

Development of DTC is continuing. Shortly, we expect that it will be able to critique decision trees by offering the analyst suggestions for correcting errors or suspected errors in structure and reasoning about uncertainty. The latter feature is expected to be especially useful since beginning analysts, in particular, often have under-developed intuition about probabilities. The development of DTC is intimately bound up with the development of the catalog of errors since we hope DTC will be alert to frequently made errors and be able to suggest corrections.

The aspect of automated decision tree construction on which Prof. Szolovits has concentrated is developing mathematical expressions for revision of context-dependent variables in a decision tree. One aspect of this is verification that a given expression gives the correct result. Another aspect is the generation of the expressions. In simple cases, this is a straightforward application of Bayes' rule. However, even experienced decision analysts are familiar with performing such Bayesian revisions only for the results of diagnostic tests. In other cases, such as observations of the clinical course of patients during a period of empiric treatment, the method is much less straightforward. Also, the classic formulation of Bayes' rule, as it is taught to physicians, considers a binary outcome such as a positive versus a negative test result. When results are not binary, the expressions, while still straightforward, become more complicated, and are less familiar to most decision analysts. The most difficult situation of all is when outcomes are not mutually exclusive. Bayes' rule cannot handle such a situation, but it is frequently encountered in applied decision analysis.

In certain cases, all of these problems can be dealt with by expanding the decision tree to the "finest grain" level, that is explicitly modeling the disaggregation of all events which can occur. For example, consider a case in which the patient may or may not have Disease X and there is an imperfect test for Disease X. There are four possible outcomes; true positive, false positive, true negative and false negative. However, the only observable event is whether or not the test is positive. If one has predetermined that patients with a positive test will be treated and patients with a negative test will not be treated, then the four terminal outcomes of the tree can be assigned in a straightforward manner.

The problem arises when the subsequent course has not been predetermined, and there is an "embedded" decision node following the test result, for example, whether to perform further testing. In this case, events cannot be disaggregated "upstream" to the embedded decision node, because the knowledge of the actual diagnosis is not available to the decision maker. All that is known is the revised probability that patient has the disease given the test result. Hence, the separation of patients into categories must be handled entirely with probability revisions and the events can be disaggregated into their terminal events only "downstream" from the embedded decision node.

In the course of performing an analysis which raised all of these issues, an algorithmic approach was developed which enabled the generation of the necessary symbolic expressions for probability revisions. The approach requires delineating an event space for the clinical situation, that is, one which considers all of the clinically relevant observable outcomes. The event space can be represented as a probability tree which represents events occurring upstream to the point where the probability revision is required. Instead of being concerned with utilities of outcomes, the focus is on counting outcomes. The way in which the event space is constructed and the probabilities used will be determined by the dependencies among various events, for example, the dependency of a test result on the presence of disease. The structure of the event space will also depend on which variable is being revised. Thus, there is not a single event space which is needed, but several event spaces, all of which represent the same information in different form. The necessary expressions are generated from the path probabilities of the events under consideration.

The need for this approach was suggested in the course of a real applied decision analysis of the strategies for evaluating pulmonary infiltrates in patients with the Acquired Immune Deficiency Syndrome (AIDS). The event space for empiric treatment in this clinical setting involves consideration of two sets of diagnoses which are not mutually exclusive, and multiple diagnoses within one set. Furthermore, the observable result during empiric therapy is not binary, but rather, involves three outcomes. The expressions required for probability revision in this tree were so complex that they could not be debugged without an algorithmic approach. This type of analysis requires far more information than we specify in a decision tree, namely dependencies among events. At the current state of the art in decision tree construction, such dependency information is provided only implicitly in the form of binding expressions, if at all. It is clear from the preceding discussion that this is somewhat backwards, because in difficult cases, an explicit specification of event dependency is required first Needless to say, in terms of communicating results, an explicit representation of dependencies is far more clear than a complicated mathematical binding expression.

A verification algorithm was devised which established that the expressions resulting

from the algorithmic approach were in fact correct. Basically, it involves establishing the probabilities of all states prior to the probability revision, and then summing the probabilities of the same states following the probability revision. The net probability of each state has to be the same both before and after the revision. If it is not, the revision is incorrect. This is equivalent to saying that performing the test and interpreting the results cannot change the state of the world. This verification algorithm would be a valuable adjunct to automated decision tree critiquing, a project which is well underway by others in our group.

Work to be undertaken in the immediate future includes submission of the results of the above-mentioned decision analysis to a major medical journal and submission of the technical aspects of the analysis to the decision analytic literature. The technical questions relate to at least three major categories; modeling empiric therapy, dealing with embedded decision nodes, and revision of probabilities when events are multiple, or not exclusive.

It seems to Prof. Szolovits at this stage of the work that the best way of specifying dependencies among events is via an influence diagram. A second major area for future work is developing a computer program which will be able to construct an influence diagram from assumptions used for the analysis and will be able to use this information to construct event spaces needed for the probability revisions. These event spaces could then be used to generate symbolic probability expressions needed to revise variables. The way in which this program would request information from the user would be an ideal way of exploring issues in developing an interface for a decision tree construction program. Such a program ideally would enable a naive user, one with no decision analytic experience, to guide the construction of a correct decision tree merely by specifying the elements of the clinical setting. The verification algorithm, while simple in principle, is computationally costly and lends itself to automated support. Another goal is to build such error checking into currently used decision tree building software.

2.8. Planning Under Uncertainty

Our view of the problem of planning patient management is strongly influenced by the domain realities of multiple, conflicting objectives in the presence of substantial uncertainty. It is not surprising, then, that our work on a therapy planner is highly related to our other efforts in using AI techniques to support decision-analytic modeling.

The planning project, part of Mr. Wellman's doctoral thesis research, considers plan construction under uncertainty to be analogous to the formulation of a decision model (a decision tree, for example). Options under consideration are initially brought into the

plan as decision nodes, and events important for prognosis or contingency planning are integrated into a probabilistic model. The plan constructor proceeds in a cycle of formulating, recognizing, and resolving tradeoffs.

Formulation consists of adding the medically relevant events and actions for consideration, driven by the problem statement presented to the planner. The program avoids generating everything in the knowledge base associated with the problem (potentially all of medicine is somehow related to any problem) by focusing on presenting details. That is, the planner only needs to consider the issues resulting from the current patient's differences from cases with known solutions.

A "tradeoff" for our purposes is a decision that cannot be resolved with purely qualitative information. For example, performing a costless informative test is not a tradeoff because there are only benefits, while a diagnostic test with possible complications does present a tradeoff. We have already developed a formalism and an algorithm by which these kinds of tradeoffs can be readily recognized within a partial decision model.

The nontrivial tradeoffs determine the focus of the next round of planning formulation effort. Tradeoff areas are worked out in greater detail so that a dominance prover can attempt to resolve the decision one way or the other. Procedures for determining dominance will be based in part on our previous work in decision making from partial utility specification. [1]

3. A PROGRAM FOR THE MANAGEMENT OF HEART FAILURE

The overall objective of this research is to develop a methodology and tools for assisting the physician in reasoning about the diagnosis and management of disease, particularly when causality, physiological relationships, and changes over time are important to the reasoning process. The context for this research is the diagnosis and management of heart failure, a problem which requires a thorough understanding of the hemodynamic and physiological relationships of the cardiovascular system for effective management. The emphasis of the early years of the project is on developing an adequate representation for the physiological knowledge, developing the appropriate physiological model for the domain, and developing the algorithms for reasoning about the diagnostic and management problems of the patient.

The participants in this project include Dr. Long and Mr. Yeh at MIT and cardiologists S. Naimi, M.G. Criscitiello, and S. Pauker at Tufts-New England Medical Center Hospital. Until October we also had a cardiology fellow, Dr. S. Kurzrok, working with us.

During the past year we have started developing software that takes advantage of the

Symbolics 3640 hardware. We have been working on physiological models that incorporate the important parameters of both sides of the heart. We have developed a new methodology for reasoning about the expected changes from a therapy. Finally, we have developed a new interface for entering the patient data.

The most important progress over the past year has been the development of methodology for reasoning about the changes expected from the application of a therapy. Once the patient data has been entered into the program and used to constrain the physiological representation of patient state and the diagnostic reasoning has produced a causal representation of the disease state of the patient, the next task is to suggest possible therapies with appropriate analysis. It is easy enough to find therapies along the causal chains with potential to correct parts of a problem, but therapies often have more than one action and even single actions have many implications. The problem is to determine what the overall effect of the therapy is likely to be, given the patient state.

The intuitively appealing strategy is to simulate the effects of the therapy. However that would require considerable quantitative knowledge of the patient parameters and the relationships between parameters. A qualitative simulation approach, on the other hand, sacrifices the ability to predict the outcome of opposing influences. Furthermore, it is difficult with any simulation approach to determine the important assumptions that produced the computed outcome.

The approach we have taken is based on the techniques of signal flow analysis. By assuming that the system will reach a stable state after a therapy is given, the problem becomes how to determine what the difference is between that stable state and the state prior to therapy. Since there are different time periods over which different parts of the cardiovascular system reach stability, it is necessary to simplify the system by assuming that effects with long time constants have no effect during short time periods. Thus, we can develop a picture of the system over different time periods by considering the system to be stable with respect to effects having more rapid action than the time period. The use of signal flow analysis also requires that the equations representing parameter relationships be linear. Since many of the relationships are inherently non-linear (e.g., the Frank-Starling relationship between end diastolic pressure and stroke volume) and many are multiplicative, it is necessary to model the system as piece-wise linear with the relationships dependent on the parameter values. With these assumptions it is possible to apply Mason's general gain formula to determine the relative changes in parameters in the model. This formula, however, obscures the reasons for the predicted changes. We have developed an alternate formulation that computes the gain incrementally from parameter to parameter, correcting for feedback each time a new feedback path is encountered.

There are several advantages to this new approach. First, since the contributions of the various pathways between parameters are explicitly represented, it is possible to explain the predictions in terms of the mechanisms most responsible for the change. This allows the user to focus attention on the more important parameter values and assumptions. Second, with the use of a number of computational techniques to increase efficiency, it is possible to compute the changes implied by a therapy on all of the parameters in the model in a few seconds, allowing effective comparison of therapies and assumptions about patient state. Third, the method only requires information about the state of parameters that will determine the strength of the links between parameters --- the determinants of the non-linear relations. Thus, the system can function without knowing all of the parameter states. Figure 2-4 shows the result of the system computing the changes expected from a beta-blocker in a patient with chronic angina. The letters at the bottom of the parameter nodes indicate the parameter states and the arrows at the top indicate the computed changes. The highlighted lines between beta state and myocardial ischemia are the three primary pathways determined by the computation accounting for the predicted decrease in tendency to have angina.

In addition to the development of this methodology for predicting change, we also have spent considerable time developing the physiological parameter model, displayed in Figure 2-4, which includes the right and left sides of the heart, the determinants of myocardial ischemia, the sympathetic reflexes, and the renal determination of blood volume. The form of the model is intended to correspond as much as possible to the reasoning of cardiologists, in keeping with the thrust of the project. The relationships between parameters in the model are formulas determining the link strengths from the parameter values. The parameter values are scaled such that experiential values of weak, moderate, and strong link values (indicated by +'s and -'s) correspond to 0.5, 1.0, and 1.5. These have proven sufficient thus far in accounting for the behavior of the therapies we have examined. To validate the model, we have compared its predictions with drug data reported in the literature with good agreement. There is still much work to be done on the model as we compare its predictions with various disease states and make additions and changes to appropriately model the diseases in the heart failure domain.

Another task of the past year has been to start moving the software originally developed on a PDP-10 over to the Symbolics 3640 Lisp Machine. Simply making the code work on the Lisp Machine was done relatively quickly, but we are still in the process of integrating the computational mechanisms into a system that takes advantage of the graphic capabilities of the hardware. The first step in this effort was to design a new input interface to take advantage of the mouse and menu capabilities of

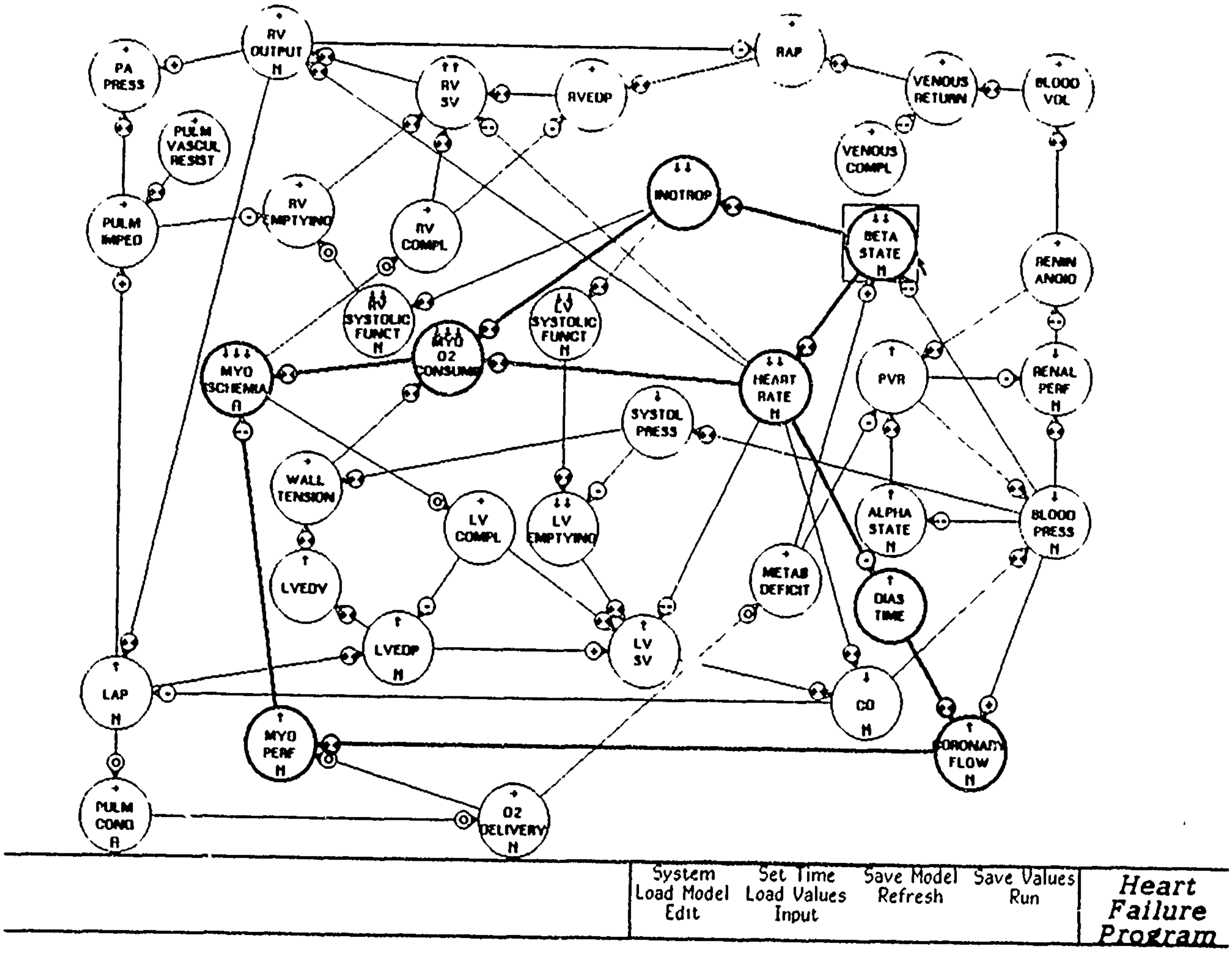

Figure 2-4: Cardiovascular Model Showing Effects of a Beta-Blocker.

the system and cut down on the need for the user to type. Part of the new input menu is displayed in Figure 2-5. The entries are divided into those with categorical values and those with numeric values. The system supports arbitrary constraints among categorical values and appropriate precision for numeric values. The system also allows new entries to pop up when particular values are touched. For example, when the user indicated that there were rales on the chest exam, the line requesting characterization of the rales was inserted in the display. The entry of numeric information is handled by providing scales along which the mouse can move to pick the desired value. As the user moves along the scale, a running value is reported to aid in finding the desired value. The layout and selection of the input information have gone through a number of refinements, culminating in the present layout. The intention is to capture the information pertinent to the heart failure without requiring the system to do

reasoning that is not pertinent to the domain. From the examination of a number of cases, it appears that the present layout captures the necessary data.

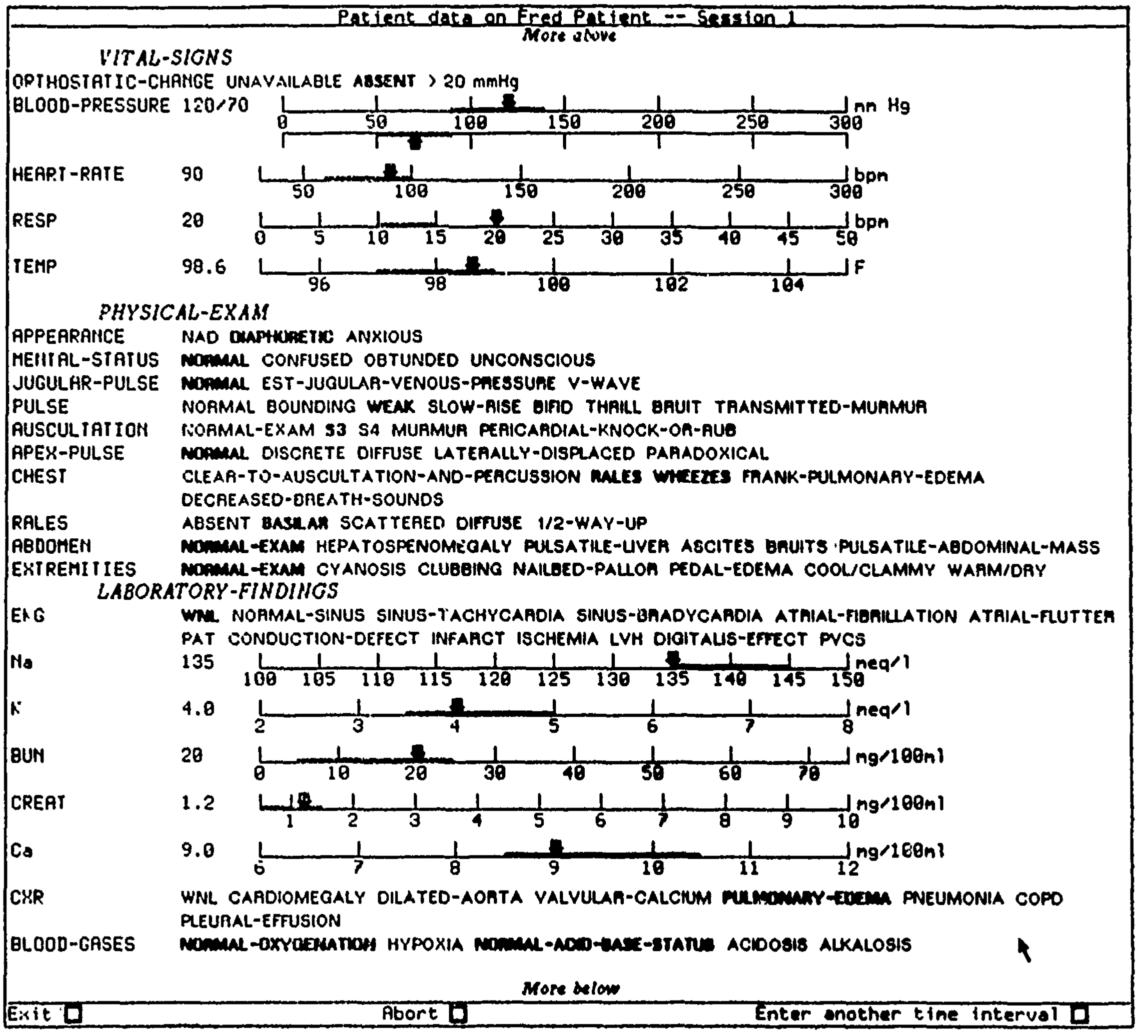

Figure 2-5: Patient Data Input Screen.

We have several objectives for the coming year. The first is to integrate the software used in the angina system with the new software developed for predicting the changes with therapy. The first step in that process was developing the new input system. The remaining steps include extending the diagnostic model to include the same parameters as the therapy model, developing evidence interpretation rules to set the parameter states from the new input list; extending the graphics system to display the diagnostic reasoning in the same manner as the therapeutic reasoning; and transferring the diagnostic capabilities.

We also intend to extend the approach used for determining parameter values in the angina system. At present, the findings are used as evidence for one or more parameter values. The evidence for each parameter value is separately computed and only those for which there is strong evidence are asserted. The logical implications of those parameter values are asserted by the truth maintenance system and the rest of the diagnostic reasoning requires assistance from the user. We plan to extend this system by providing a probabilistic (or pseudo-probabilistic) mechanism for handling relationships between parameter values that are less than definite. We have examined various schemes for handling such relationships and find that the work on Bayesian networks by J. Pearl and others is almost what we need to extend our system. There are some problems that will have to be overcome, since our network does have loops in it and the large number of splits and joins in the network may lead to computational problems, but the techniques show promise.

There is still a considerable amount of work to be done in refining the physiological model as used for therapy prediction. We need to extend it to cover the disease states that cause or complicate heart failure in the adult. We will also continue the analysis of the model to better understand the relationship between this type of model and the more traditional physiological simulation models.

One major goal for the middle part of 1986 is to have the system ready for initial use by students and staff in the CCU. This will mean developing data base facilities to keep and recall the data they enter, a recording and comment facility to enable examination and analysis of the use of the system, online documentation available from any part of the program, and a general tying up of the many loose ends that are currently in the system.

References

1. Wellman, M. "Reasoning About Preference Models," MIT/LCS/TR-340, MIT Laboratory for Computer Science, Cambridge, MA, May 1985.

Publications

1. Koton, P. "Empirical and Model-Based Reasoning in Expert Systems," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-85)*, University of California, Los Angeles, CA, August 1985, 297-299.
2. Patil, R. "Coordinating Clinical and Pathophysiologic Knowledge for Medical Diagnosis," *Proceedings of the Seventh National Conference of the American Association for Medical Systems and Informatics (AAMSI)*, Anaheim, CA, May 1986, 3-8.
3. Sacks, E. "Qualitative Mathematical Reasoning," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-85)*, University of California, Los Angeles, CA, August 1985, 137-139.
4. Szolovits, P. "Knowledge-Based Systems: A Survey," *On Knowledge-Base Management Systems: Integrating Artificial Intelligence and Database Technologies*, M.L. Brodie and J. Mylopoulos (eds.), Springer-Verlag, 1986, 339-352.
5. Szolovits, P. "Types of Knowledge as Bases for Reasoning in Medical AI Programs," *Artificial Intelligence in Medicine*, I. De Lotto and M. Stefanelli (eds.), Elsevier Science Publishers, B.V., North-Holland, 1985, 209-226.
6. Wellman, M. "Reasoning About Assumptions Underlying Mathematical Models," *Coupling Symbolic and Numeric Computing in Expert Systems*, J.S. Kowalik (ed.), North-Holland, 1986, 21-35.
7. Yeh A. "Flexible Data Fusion (& Fission)," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-85)*, Los Angeles, CA, August 1985, 420-422.

Theses Completed

1. Hirsch, D. "An Expert System for Diagnosing Gait in Cerebral Palsy Patients," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.
2. Smit D. "Knowledge-Based Interpretation of Elevated Roadway Condition Surveys," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February 1986.

Theses in Progress

1. Goldberger, H. "DAISY: A Model for Temporal Reasoning in Medicine," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected June 1987.

Talks

1. Goldberger, H. "The Medit-Medic System: Tchad," *International Workshop on Medical Expert Systems and Social-Economic Opportunities for Their Implementation in Developing Countries*, Laxenburg, Austria, November 1985.
2. Koton, P. "Empirical and Model-Based Reasoning in Expert Systems," *International Joint Conference on Artificial Intelligence (IJCAI-85)*, University of California, Los Angeles, CA, August 1985.
3. Patil, R. "An Analysis of Hypothetico-Deductive Reasoning in Medical Diagnosis Programs," Knowledge Based Applications Course, Harvard University, Cambridge, MA, November 1985.
4. Patil, R. "Evolution of Diagnostic Expertise in AI in Medicine Programs," IBM, Yorktown Heights, NY, November 1985.
5. Patil, R. "A Case Study in Evolution of System Building Expertise: Medical Diagnosis," MIT Symposium on *Artificial Intelligence: Current Applications, Trends and Future Opportunities*, sponsored by Artificial Intelligence Laboratory/Center for Advanced Engineering Studies, Cambridge, MA, January 1986.
6. Patil, R. "Multilevel Representation of Complex Medical Knowledge," *Harvard Seminar in Medical Information Science*, Boston, MA, February 1986.
7. Patil, R. "Organizing Medical Knowledge for Efficient Diagnosis," keynote speaker at *Third Annual Computer Science Symposium on Knowledge Based Systems: Theory and Applications,* University of South Carolina, Columbia, SC, April 1986.
8. Patil, R. "Coordinating Clinical and Pathophysiologic Reasoning for Medical Diagnosis," Award Lecture, Young Investigator Award for Research in Medical Knowledge Systems, *AAMSI Congress 1986*, Anaheim, CA, May 1986.
9. Sacks, E. "Qualitative Mathematical Reasoning," *International Joint Conference on Artificial Intelligence (IJCAI-85)*, University of California, Los Angeles, CA, August 1985.
10. Szolovits, P. "Computer Science and Decision Support," *Conference on the Medical Information Sciences,* University of Texas Health Science Center, San Antonio, Texas, July 1985.

11. Szolovits, P. "Expert Systems: Introduction," tutorial at *International Joint Conference on Artificial Intelligence (IJCAI-85)*, University of California, Los Angeles, CA, August 1985.

12. Szolovits, P. "Expert Systems: State of the Art," tutorial at *International Joint Conference on Artificial Intelligence (IJCAI-85)*, University of California, Los Angeles, CA, August 1985.

13. Szolovits, P. "Structural Representation of Uncertainty," Panel presentation, *International Joint Conference on Artificial Intelligence (IJCAI-85)*, University of California, Los Angeles, CA, August 1985.

14. Szolovits, P. "Types of Knowledge as Bases for Reasoning in Medical AI Programs," *International Conference on Artificial Intelligence*, Pavia, Italy, September 1985.

15. Szolovits, P. "Prospects for Practical Applications of AI in Medicine," Panel presentation, *International Conference on Artificial Intelligence*, Pavia, Italy, September 1985.

16. Szolovits, P. "Medical Artificial Intelligence Research in the United States," chief invited address, *International Conference on Artificial Intelligence in Medicine and Biology*, Tokyo, Japan, January 1986.

17. Szolovits, P. "Expert System Tools and Techniques: Past, Present and Future," MIT Symposium on *Artificial Intelligence: Current Applications, Trends and Future Opportunities*, sponsored by Artificial Intelligence Laboratory/Center for Advanced Engineering Study, Cambridge, MA, January 1986.

18. Yeh, A. "Flexible Data Fusion (& Fission)," *International Joint Conference on Artificial Intelligence (IJCAI-85)*, Los Angeles, CA, August 1985.

COMMON SYSTEM

Academic Staff

D. Clark, Leader
D. Gifford
B. Liskov
S. Ward
W. Weihl
R. Zippel

Research Staff

T. Bloom
I. Greif
R. Scheifler
K. Sollins

Graduate Student

S. Zanarotti

Support Staff

J. Doherty
G. Staton

Visitor

J. Emer

1. OVERVIEW

During the current year, we have started a new distributed systems project. This project, the LCS Common System Project, has the goal of understanding how programs written in different languages on different machines can invoke each other. The motivation for this project is the observation that, both locally and in the world at large, the number and variety of computing systems is increasing steadily. The incompatible programming environments on these various machines create substantial barriers to program interoperability. Within our community, we perceive serious problems in building on the work of others, because the different program development environments used within the Laboratory effectively partition us into isolated groups of programmers.

The objective of this project is thus to permit a program to be invoked uniformly from within all of our programming environments, independent of the particular language in which that program was written. Within the Laboratory today, we see a number of programming languages with vastly different characteristics: Lisp, with its rich runtime environment; Clu and Argus, with their strong compile time type checking; and C, with comparatively primitive semantics and development tools. Our specific goal is thus to permit program invocation among any of these three language communities.

As a practical matter, if we are to provide universal program access, we must solve two distinct problems; the invocation of a program by another program, and the invocation of a program by a person at a display. In the one case, we must provide an invocation mechanism which meshes together the distinctive semantics of the various languages. In the case of human invocation, we must provide a model of a display which masks the detailed physical characteristics of the display being used by that particular person. In both cases, the issue involves the definition of high level standards and abstractions. It is the definition of these standards which lies at the heart of the project.

2. REMOTE INVOCATION

There are a number of ways in which a program written in one environment could be made usable from another environment. For example, automatic translators could be proposed, which would take a Lisp program and automatically produce an Argus or C program with the same function. We reject this approach as totally unworkable. We believe that, as a practical matter, programs must continue to execute within the native environment in which they were developed. Since the various languages typically execute on different hardware bases, cross language program invocation thus implies invocation over a computing network. While the central focus of our research is multi-

language invocation, rather than distributed invocation, the distributed aspect of our languages has substantial performance implications for our overall design. In fact, the distributed aspect of our system turns out to be almost as important as the multiple languages in the impact on the system semantics.

At the heart of our system is an *invocation model*, which specifies the manner in which one program may call another. There are several possible paradigms for invocation: procedure call, co-routine, or message sending between parallel asynchronous processes. We prefer the procedure call model, in which the calling program suspends its execution at the time that it invokes the sub-routine, and waits for the return values before continuing. This model matches the vast majority of inter-program communication today, is relatively easy to understand, and compatible with the calling semantics of almost all programming languages. However, the distributed nature of our invocation mechanism raises performance problems which require a more complex invocation paradigm. Based on the applications we have studied, we believe that we must support, in addition to remote procedure call, the sending of reliable sequenced one-way messages. Since one message can be sent without waiting for a previous reply, the rate of message sending is no longer limited by the round trip delay across the network, as it would be in the case of a remote procedure call. Instead, messages can simply be sent as fast as the originator can generate them and the receiver can process them. For some applications, such as the remote updating of a display, this reduction in latency is critical to an acceptable level of performance. Our invocation paradigm is thus remote procedure call augmented by reliable streams of messages.

When a program is to be exported within the Common System, so that it [illegible] invoked remotely, the Common System must construct a representation [illegible] program, which can be passed around within the system. We call this abst[illegible] *port*. Formally, a port has the following components: the location of the service (e.g., host address), the types of the arguments of the procedure, the types of the return values of the procedure, and any exceptions or error conditions which the procedure may return in place of its normal return values. As a practical matter, the type information will be encoded in some efficient way, such as an index into some catalog of definitions.

Our invocation model recognizes that it is important to gather groups of ports together into an aggregate element. The most obvious reason for gathering together a number of ports is to collect into one abstraction all of the operations performed by a particular service. For example, a file system might have separate ports for opening a file, reading to and writing from a file, and closing a file. It is useful to pass around this set of ports as a unit. In fact, many representations of remote procedure invocation use the word "port" to describe this aggregate, and use the word "operation" to describe the

individual entry points of the service. (Although there are some distinctions between the two representations, these do not appear to be of first importance.) Another reason for gathering ports together into an aggregate is to identify a set of ports among which the messages and procedure invocations are to be performed sequentially. If our invocation semantics were a pure remote procedure call, this concept would not be important, because procedure calls are by definition sequenced; the client cannot make a second call until the first one has returned. However, once we permit messages without replies, then we must think more carefully about sequencing. It is not sufficient to say that the messages sent to one particular port must be sequenced; it may also be important that messages sent to two ports arrive at those ports in the order in which the client sent them. Imagine, for example, a program to manage a display, which exports two sequential message ports, one to write a character to the display and the other to change the character font. Messages to those two ports must be properly sequenced if the characters are to be written to the display in the font intended by the client. Our mechanism must thus support a mechanism for permitting several message streams to be collectively sequenced, and this can best be represented by gathering together those ports into some sort of aggregate structure.

We believe that arguments and return values from remote procedures are passed by value. That is, our system would normally pass the object itself, rather than some sort of referent or descriptor for the object. In this respect, our system differs from a classical object-oriented distributed system, in which the objects along with their operations are stored inside some site. In our system, we imagine data values being passed from one machine to another, and manipulated at various sites using the native programs which are available on that machine. We believe that the correct way to characterize the values that are passed between procedures is to define a basic set of data types and constructors (e.g., array and record) which can be combined together to characterize the data type of each value. These sorts of data types are not precisely the same as those which are traditionally used in typed data languages, because one typically thinks of a data type as being characterized by the operations which can be performed upon it, as opposed to its representation. For example, an integer is characterized by the fact that it supports the traditional arithmetic operations, not by the fact that it has a certain number of bits. In our system, however, we are not in a position to constrain the operations which can be performed on the data item. We are thus reduced to a primitive representation for type, which characterizes the data item by the range of values which it can assume. To distinguish this form of type from other uses of the word, we have coined the word *c-type*.

An essential component of the Common System is a *catalog*, which records information about all of the programs that are available within the system. In particular,

the catalog must record the c-type of all of the arguments and return values for every port. The port value itself probably contains only an index to the catalog entry for the collection of c-types. For those ports which are to be widely used over a long period of time, the catalog will also provide a place for storing the port values away and retrieving them based on some naming structure, such as a hierarchical name-space.

The final aspect of invocation semantics which we have designed to this point has to do with the semantics of failures. It is generally believed that failures in distributed systems are more likely than failures in centralized systems, due to the unreliability of communications. Certainly, failures in distributed systems are harder to recover from, because the damage that results from a failure occurs in an environment other than the one in which the programmer is familiar. If manual reconstruction of state is necessary, it may be very difficult for the programmer to master the necessary skills. For this reason, we feel that it is very important to constrain the class of failures which can occur as a part of invocation through the Common System. Our approach to control of failures is to include within the Common System a coordinator for atomic transactions. This means that services will be able to structure themselves around a transaction model, in which operations either commit, in which case all of the results are visible, or abort, in which case, none of the results are visible. The failure modes are thus very simple to explain; a program either runs or does not, and there are no intermediate results which may become visible as a result of an error.

Most of the language environments which participate in the Common System do not have a native facility to implement atomic actions. However, we believe that by providing a file system which provides atomic updates, we can capture most of the important characteristics of a fully implemented transaction system without modifying the native environments of those systems which do not already support this capability.

3. PROGRAM SPECIFICATION

Several steps are required in order to make a program written in one of the native languages available through the Common System. First, of course, it is necessary that the program be constructed taking into account capabilities which the Common System provides. If the program requires an invocation model which is alien to the Common System, then it will be impossible to market the program through those interfaces. The second step in installing a program in the Common System is to describe the interface to the program in such a way that the Common System can construct a port value for it. For this purpose, the Common System will provide a specification language in which one describes the various ports which should exist for a service, and the types of the argument and return values. The Common System will use the specification in a

number of ways. First, of course, the specification will permit the Common System to construct a valid set of port values for the program. Second, the Common System will use the specification to generate, in each of the languages native to the Common System, an *invocation stub*. The invocation stub is the manifestation of this program in each of the native languages in which the program is to be called. The invocation stub will automatically prepare the arguments in the correct format, starting with values stored in the representation of the native format. The Common System thus attempts to provide the illusion that this remote program, written in some other language, is in fact not very different from a local sub-routine written in the language of the caller. The various specifications and invocation stubs are presumably stored in the catalog, along with the port values and type definitions of the interface. It should thus be obvious that the catalog is a fairly complex mechanism, central to the operation of the Common System.

4. SERVICE INTERFACES

The Common System, as described so far, contains a definition of invocation semantics, a specification language, and a catalog. In fact, there are several other important components to the Common System. In particular, there are certain services which are central to the Common System, whose interface must be defined, using the components of the Common System described so far. Some of these interfaces are fairly obvious, for example, an interface to an atomic file system. A very important interface, which we have not mentioned so far, is the interface to the display and input devices of the human overseeing the computation. Within the Common System, we assume that there will be several different computer workstations used as interface devices for humans. At the present time, we are using Micro-Vaxes with a variety of high resolution displays, Symbolics Lisp Machines, and TI Explorer Lisp Machines. These machines differ, both in the physical details of their display, and in the low level software provided for manipulation of the display. Essentially, all these have in common is that they provide a high resolution all points addressable multi-window display. As part of the Common System, therefore, it is necessary to define an abstract view of this class of display which is capable of being mapped on to any one of these physical devices.

At a low level, such a definition already exists: the X-Window System developed by R. Scheifler of this laboratory and described elsewhere in the annual report. X provides remote access to a rectangle of pixels on a display, and supports various manipulations on this rectangle, such as displaying text or performing graphic actions. However, X is a very low level interface, and does not address such high level issues as the user interface for window management or command input. We have not yet addressed the

question of how the Common System should deal with this higher level interface, but this remains an important problem for future research.

Another interface which we wish to specify as a standard within the Common System is an interface to a centralized authentication server. It is undesirable for each machine within the Common System to maintain its own independent model of user authentication. It is much more desirable if services marketed through the Common System can turn to a common authentication service to validate the identity of their clients. In order that this work smoothly, it is necessary that the interface to the authentication service be defined early.

5. FUTURE PLANS

The Common System is conceived as a fairly large project which will involve a number of faculty members for the next several years. This first year has been spent in high level planning, and in specification of basic components of the system, such as the invocation model. During the next six months, we expect to have a set of design documents sufficient for us to begin early experimentation with remote invocation. We expect the Common System to become available incrementally over the next two to three years.

COMPUTATION STRUCTURES

Academic Staff

Arvind, Group Leader

J.B. Dennis

R.S. Nikhil

Research Staff

W.B. Ackerman
G.A. Boughton
R.A. Iannucci
J.T. Pinkerton

Graduate Students

M.J. Beckerle
S.A. Brobst
A.A. Chien
T-A. Chu
D.E. Culler
G-R. Gao
B. Guharoy
S.K. Heller
R.A. Iannucci
S. Jagannathan
B.C. Kuszmaul
V.K. Kathail
G.K. Maa
G.M. Papadopoulos
K.K. Pingali
S.A. Plotkin
R.M. Soley
K. Theobald
K.R. Traub
B. Vafa
E. Waldin

Undergraduate Students

D. Anderson
A. Chang
S-W. Chen
D. Clarke
B. DeCleene
S. Kaushik
S. Ma
D. Marcovitz
D. Morais
M. Ng

S. Desai
C. Goldman
E. Gornish
R. Griffith
R. Gruia
E. Hao
J. Hicks
F. Herrmann
R. Rabines
P. Rosenblum
S. Sanghani
J. Soong
Y-M. Tan
S. Younis
C. Winters

Support Staff

S.M. Hardy
N. F. Tarbet

Visitors

M. D. Atkins
B. Blaner
M. Mack
H. Nohmi
N. Skoglund
S. Truve

1. INTRODUCTION

The Functional Languages and Architectures Group (FLA) was started in January 1981, splitting off from the Computation Structures Group (CSG) to continue research in tagged-token dataflow architectures and functional languages. During this past year, the old CSG was disbanded when its group leader, Professor J. Dennis, began a leave of absence. Professor Dennis and the remaining members of the old CSG joined the FLA group, led by Professor Arvind. FLA has taken on the name of its predecessor, and thus the group organization has been restored to what it was in 1980.

A major thrust of the new Computation Structures Group is in two interrelated projects -- the Multiprocessor Emulation Facility (MEF) and the Tagged-Token Dataflow Machine. The goal of the MEF project is to construct a "sandbox" to facilitate research and development in parallel architectures and languages. The most exciting development of the year has been the execution of large dataflow programs on the 32-processor MEF. Though much work is continuing on the MEF front, the group is concentrating on dataflow research, now that a reliable MEF foundation is in operation.

The goal of the Tagged-Token Dataflow project is to demonstrate the feasibility of general-purpose parallel machines by simulation and elation. In the past year we have developed some major tools for dataflow research, enabling us to conduct several experiments with large programs, significantly improving our understanding of dataflow architectures.

We have also implemented a preliminary version of a Parallel Graph Reduction Machine on the MEF. We hope to solidify the prototype and use it to study these architectures in the coming year.

As part of the Tagged-Token Dataflow project, we have also been looking at several issues related to language design and compilation, such as data structures, data types and storage management, and the relationship and tradeoffs between data-driven and demand-driven computation.

Professor Nikhil has begun studying the integration of databases into functional languages, based on a model that has a rich type structure and explicitly manages histories of states.

In addition, several of Professor Dennis's students are pursuing different research goals. B. Kuszmaul has been exploring the simulation of applicative languages on the Connection Machine [16]. G-R. Gao continues to investigate compilation of VAL into efficient code for the static Dataflow Machine [9].

IBM has continued to support a small group of engineers assigned to CSG whose function is to assist in the design and development of the packet switch. This group has

demonstrated the first working pieces of the packet switch (the crossbar and the scheduler/arbiter) using a cMOS gate array and MSI TTL components. The group has also refined and verified the serial link protocol for the packet switch, and is well on the way to releasing a 10,000 gate array which performs the serial receiver and transmitter functions, as well as FIFO buffer control.

As a consequence of the MIT-IBM technical review meeting in the summer of 1985, CSG initiated and hosted a series of seminars by prominent IBM people. The series, entitled "Large and Complex Computer Systems in the Commercial World," was designed to give members of the LCS community a view of issues and problems in the design of large-systems in the "real" world.

I. Wladawsky-Berger, Vice-President of Development for IBM's Data Systems Division, began the series with a talk entitled "Trends in Large Computer Systems" in which he discussed technical problems in the growth of large systems. In December, T.W. Scrutchin, technical assistant to the manager of the Transaction Processing Facility, discussed the application of this system to airline reservations and banking systems. Federal systems division representative K.Y. Rone, a 20-year veteran of IBM's On-board Space Systems Organization, came in February to present an overview of the computer systems research that has gone into the space transportation aspects of the nation's aerospace program. Concluding the series in April, K.R. Milliken, a project manager at the IBM T.J. Watson Research Center, talked about YES/MVS, a program which applies AI expert systems techniques to assist operators of large MVS installations. Judging from the size of the audiences and from the lively discussions that followed each talk, the series appealed to a wide cross-section of the Laboratory.

The group has decided to write all its new programs in Common Lisp, the new standard dialect of Lisp. Through the efforts of R. Soley, the group continues to have strong ties to the Common Lisp standardization effort, both in the old Common Lisp committee, and the ANSI X3J13 and ISO TC97/SC22 Lisp standardization committees.

2. PERSONNEL

We were fortunate to have Dr. G.A. Boughton, a member of the old Computation Structures Group, join the new CSG as a research associate, assuming responsibility for the welfare of the MEF, in particular the circuit switch. Dr. W. Ackerman, another member of the old CSG, came on board at the same time, but left for Apollo in February. He was a great help in getting all the new Texas Instrument Explorers (Lisp Machines) up and running. N. Skoglund, a visitor from Ellemtel, has returned to Sweden. During his stay, he developed the NuBus interface logic for the NuCA (NuBus Channel Adapter), and began construction of a wire-wrapped prototype. The bulk of the NuCA will be tested through incremental additions to this prototype.

The IBM group that has been working with us for the past several years has also changed. M. Atkins, a Senior Associate Engineer from Endicott, joined the group in November and has been working on the design of the 48 MHz clock subsystem and the FIFO control logic. At the end of April, B. Blaner returned to Endicott. During his stay, he completed work on the serial protocol and has also completed detailed design on 80% of the serial receiver logic.

3. MULTIPROCESSOR EMULATION FACILITY

This past year has been one of successes for the Multiprocessor Emulation Facility (MEF). Thirty-eight Texas Instruments processors are now on site, with 32 fully operational as part of the facility, connected via the circuit switch as well as standard Ethernet. The eight existing Symbolic machines are now used primarily for software development. The old MEF software base, designed and implemented by R. Soley, was retired this past year, making way for the new Id World MEF software supporting the new circuit switch.

3.1. The Current MEF Hardware and the Circuit Switch

During the past year, we have taken delivery of 38 Texas Instruments Explorer Lisp Machines. Each Explorer is equipped with eight megabytes of physical memory, 140 megabytes of disk storage, ethernet, and bit-mapped display. Thirty-two of these machines are connected by a circuit switching network. The circuit switch is based on the Butterfly switch protocols of Bolt, Beranek, and Newman (BBN). The basic protocol has been robustified with the addition of routing header encoding, data parity, and error signalling.

Every TI Explorer hosts a circuit switch node. Each node consists of four input links and four output links. Any input can be routed to any free output through a 4 x 4 crossbar. One input link and one output link is dedicated to the host processor, while the remaining three input and output links can be connected to other nodes to form an arbitrary network topology. We have chosen a modified hypercube, called a *directed hypercube,* as the MEF network [12]. Developed by S. Heller, the directed hypercube is similar to traditional hypercube networks except that edges are directed; data can only flow in the direction of the edge. This enables the network to accommodate 64 machines in a six-dimensional directed hypercube rather than the maximum of eight processors that could be achieved using a traditional hypercube.

Message routing is *source based.* The host processor appends a string of routing instructions to the head of each message. A routing instruction is an encoded designator of one of the four output links at each node. The node strips off the leading

routing instruction and forwards the message along the requested link. If the requested link is already active with another message, then the request is denied and the message is rejected a condition which is detected by the source processor. The source processor is responsible for retransmission of a rejected message. Although the hypercube topology makes analytic derivation of routing strings straightforward, any real implementation is subject to failed links and nodes. S. Heller and D. Culler have developed distributed algorithms that establish the topology of the network and then derive shortest path routes without any preknowledge of the current topology. The algorithm is based first on a *network flooding*, in which each processor broadcasts to all its neighbors every new aspect of the network it has discovered until no new information is derived.

The links that are dedicated to the host processor (one input and one output) are interfaced through FIFO buffering to the NuBus. These FIFOs, 2K bytes in each direction, not only perform synchronization but also word width and protocol translation to and from the 4-bit 6Mhz circuit switch and the 32-bit 10Mhz NuBus. Additional logic is supplied for message disposition reporting, retransmission, and instrumentation. The circuit switch node is implemented on a single NuBus card. The approximately 200 TTL integrated circuits are interconnected using Multiwire technology.

Each external link consists of seven signals in the forward direction: a four bit data path, a message frame signal, data parity, and data/address demarcation. The reverse path consists of two status signals, message reject and parity error, for a total of nine signals per link. Each signal is driven NRZ through twisted pair and is differentially received through optoisolators.

The three input and three output links from each machine are not directly connected to other machines to form the hypercube network. Instead, these links are bundled into twisted-pair flat cables, three links per cable, and are brought to *cluster centers* that form three dimensional hypercubes (eight processors). The cluster centers provide "dimension links" to hook together up to eight clusters, for a total of 64 machines. Developed by G.A. Boughton, the cluster centers also perform local clock distribution and supply unique node identifiers that can be sampled by any circuit switch card.

The system is globally synchronous at a rate up to six megahertz, yielding a raw per link bandwidth of three megabytes per second. The clocks are distributed by a network designed by S. Younis [24]. The distribution system, implemented on a single NuBus card and hosted by a Lisp Machine, is autocalibrating and is able to hold a 10 nanosecond phase skew across the entire facility (approximately 10 meters).

The circuit switch is in active use for sending tokens between machines in the dataflow experiments being hosted on the facility. The network has also proven

invaluable for facility maintenance. D. Morais has implemented a disk transfer protocol that can simultaneously copy a 30 megabyte Lisp World file to all 32 machines in under four minutes.

G. Papadopoulos was primarily responsible for the design and prototype construction. G.A. Boughton was responsible for the production, test, and installation of 35 boards now in active use. Additional help was provided by G. Bromley and S. Sanghani. M. Bromley implemented and tested the modified BBN protocols. Ms. Sanghani aided in schematic capture and design verification.

3.2. The Packet Switch Development

The success of the circuit switch as an interim networking solution for the MEF has given us an opportunity to develop more rigorously and verify the design of our packet switching network card. The packet switch is intended to provide two orders of magnitude more usable bandwidth than the circuit switch while significantly increasing the reliability of the net itself. The packet switch provides, on a single card, an 8x8 crossbar, routing address translation logic, and serial $\leftrightarrow$ parallel conversion circuitry. Serial links interconnect sister packet switch cards. Using one such card per Lisp Machine, networks of various topologies can be composed. R. Iannucci has been responsible for the high-level design of the packet switch [13] and has coordinated the implementation work as well. During the last year, we settled on a gate array technology for the switch and made significant progress on the design.

Our switchover to LSI Logic Corporation as our gate array foundry has been largely successful. We have fabricated our first chip -- an 8x8 4 bit wide crossbar slice on a 2200 gate, 3μ m 2 layer metal array. M. Mack designed and simulated this chip prior to manufacture on our Apollo/Mentor workstations. We received first silicon in February, and are just completing our acceptance testing. We are happy to report that the design is 100% operational and bug-free. Based on a lotsize of 10 chips (all from the same run), we find the AC performance meets the advertised worst-case specifications and, in some cases, exceeds the specs. The chip was designed with fairly wide margins and, consequently, can be used at speeds approaching 10 megabytes per second per port, while the current Packet Switch architecture only requires four megabytes per second per port.

Our second chip (called "PASS" -- a not-too-clever acronym for <u>PA</u>cket <u>S</u>witch <u>S</u>lice) is in detailed design now, and will be fabricated on a 10,000 gate 1.5μ m 2 layer metal array. The experience with the crossbar chip provides us with a data point which confirms the manufacturer's claim that if a design works under the timing simulator, it will also work in silicon. This has boosted our confidence in the PASS design.

The PASS chip is made up of three subsystems: the serial receiver, the serial transmitter, and the FIFO controller. B. Blaner has made progress in partitioning the design along these lines, and has also taken the lead in developing the receiver section. He has proposed, developed, and analyzed a D.C. balanced serial transmission code which translates each four data bits into six baud, a so-called 4b/6b code. This is an improvement over the more traditional Manchester-style D.C. balanced codes which translate each bit into two baud (1b/2b code). He has also developed and verified the serial link protocol which will be used for flow control and error recovery on the serial links. The scheme is based on delimiting at message boundaries, and streaming i.e., without flow control, within a message. B. Blaner and S. Kaushik designed and implemented a 32 bit (Ethernet polynomial) CRC generator / checker circuit for the receiver.

M. Atkins has designed and begun to implement the PASS logic for the FIFO Controller. The controller provides the next in / next out pointers for static RAM control in addition to (1) message delimitation allowing last message retransmission, and (2) instrumentation for measuring FIFO occupancy. He has also completed the high-level design of the clock logic both on-card and within the PASS chip.

M. Mack has begun design work on the transmitter. He and A. Chang have also completed the design and TTL prototyping of our scheduler subsystem as well. A mid-course redesign resulted in simplification of the clocking and interfacing of the scheduler. This resulted in a 20% reduction in chip count as well. Preliminary work has already been done to interface the crossbar chip to the scheduler prototype to test the interface.

We have come to an agreement in principle with IBM on the use of one of their internally developed high-speed bipolar phaselock loop chips as the serial line front end for each inter-switch link. J. Pinkerton has studied the analog portions of this serial link and has developed (1) a Multiwire test card to study the impedance characteristics of on-card wire, (2) a printed circuit jig and test circuit to characterize the IBM phaselock loop chip, and (3) a specification for our custom serial cable. Most noteworthy is the cable which represents the state of the art in shielded twisted pair interconnection. We are seeking price quotations now and will select a vendor shortly. One manufacturer has produced a sample of our cable which upon cursory analysis comes close to meeting our specifications.

We have also reached a decision regarding our strategic direction for circuit card fabrication. The circuit switch was designed for fabrication in Multiwire technology rather than in printed circuit for reasons of (1) simplicity in layout, (2) ease of change, (3) quick turnaround, and (4) economy for production quantities of 35 or so. Our

experience (and experience of others) has demonstrated to us that we were misguided in expecting quick turnaround and economy over printed circuit boards of equivalent complexity. These observations, coupled with the closing of the Multiwire fabrication plant in New Hampshire, forced us to reconsider the use of printed wiring.

3.3. The Hardware Laboratory and New Equipment

We approached IBM about our need for printed circuit design tools and asked that they donate to us a P.C. layout facility. They have agreed, and we are in the process of installing four IBM 5080 color graphics workstations which, when connected to the previously loaned 4381 host processor, will run the Circuit Board Design System (CBDS) developed by Bell Northern Research and marketed by IBM. We intend to use this tool to implement circuit cards as simple as two-layer, one-of-a-kind test fixtures up through an eight- or ten-layer high density card for the packet switch. This facility should be operational by August 1986.

We have installed and have started using the equipment donated by Apollo (reported last period) as schematic capture and simulation stations. We now have four workstations on our design LAN (interfaced to the DARPA Internet) along with 1.5 GB of file space and a 6250 bpi streaming tape drive.

To support our LSI Logic work, we have installed their design verifier and Software Data Book packages on these workstations. This provides us with a library of macrocells (NANDs, NORs, SSI-type structures, I/O pads, etc.,) and a set of tools for analyzing the manufacturability, timing, and loading characteristics of our gate array designs. The design verifier is usable with LSI Logic's LL3000, LL5000, LL7000, and LL9000 families, spanning the gap from 3μ m low density cMOS to 1.5μ m 10,000 gate cMOS.

We have made some progress in exercising our lab instruments through the IEEE 488 bus. We have developed a spectrum analyzer package for the IBM PC which interfaces to our Tektronix 7854 Digitizing Mainframe. When completed, this package will facilitate the analysis of the serial link section of the packet switch in addition to serving as a general-purpose spectrum analyzer.

4. TOOLS FOR DATAFLOW EXPERIMENTS

Our goal of investigating the feasibility of general-purpose parallel machines has necessitated developing a variety of tools, including a compiler for generating dataflow graphs from programs written in Id, vehicles for executing these graphs, and an environment to facilitate application development (editing, compiling, debugging) and

architectural study. The first version of the compiler is fully operational, as is a reasonably efficient graph interpreter. The Id World environment provides an Id mode within ZMACS, similar to Lisp mode, to allow interactive editing, compilation, execution, and debugging. Given a working dataflow program, we want to investigate its behavior when executed on a dataflow machine. To this end we have developed three vehicles for modeling the execution of dataflow programs on hypothetical dataflow architectures. The first provides an extremely abstract model, similar to the U-interpreter, with unit operation costs, unbounded processing power, zero communication delay, and ideal program distribution. This provides a notion of the *inherent* properties of the program. The second vehicle is a complex of 32 machines, each pretending to be a dataflow processing element and communicating by passing tokens. No statement is made about the internal structure of the processing element, however, resource management issues, such as how work and data are distributed over the complex, are addressed in earnest. An important ancillary benefit is that we really get to indulge in the *multiprocessor experience*, keep a large number of machines alive and doing what they are supposed to. The third vehicle, a detailed simulator running on an IBM mainframe, allows us to look down inside the processor, in addition to observing system-level resource management concerns.

4.1. Id Compiler

Version 1 of the Id Compiler [11], which is being maintained by K. Traub, continues to run. Several small modifications have been made in the past year, including a new schema for loop constructs which bounds the resources required to execute loops. While Version 1 has served the needs of the group adequately up to the present time, its limitations have become more and more apparent as work on the dataflow project progresses. Besides the issue of compiler robustness, the need for compiler-related experiments -- program transformations, compile-time type checking, optimizations, architectural changes, and the like -- has sharply increased.

To help accommodate experiments in program transformation, an alternative syntax for Id, called PID, has been developed. PID code strongly resembles Lisp, and as a result is easily manipulated by programs written in Lisp. An undergraduate, E. Hao, has written a program which translates from PID to Id, so that PID programs may be compiled by the Id compiler and subsequently run on the simulator or GITA [10]. Together with a modification to the Id compiler which permits translation from Id to PID, his translator allows an Id program to be translated to PID, subjected to a source-to-source transformation by a Lisp programs, and finally translated back into Id for compilation and execution. S. Heller has demonstrated the power of this simple technique by implementing the program transformation described by K. Pingali which

causes a program to be executed in a demand-driven manner on a data-driven machine [22].

While the PID-to-Id translator was a quick way of enhancing the power of Version 1 of the Id compiler, the need still exists for a compiler designed to accommodate all manner of compiler-related experiments. Version 2 of the Id compiler is currently under development, and is specifically designed to support experimentation. Its major design goals:

- Provide a common core of data structures and abstractions that is general and powerful enough to support all conceivable dataflow compilers. This common core is described in K. Traub's "A Dataflow Compiler Substrate" [23].
- Provide a well-documented modular structure that allows the easy addition and removal of compiler phases, such as type-checking or common subexpression elimination. In this way, experimental compilation techniques can be introduced at each step of the compilation procedure.
- Make the major transformation phases of the compiler (parsing, program graph generation, machine graph generation, and assembly) as specification-driven as possible. This allows easy modifications to syntax, schemata, and machine architecture.
- Implementation in Common Lisp, to insure portability.

Development of Version 2 is well under way, and is expected to compile its first program in August 1986.

4.2. GITA

Last summer a new vehicle, GITA (for Graph Interpreter for the Tagged-Token Architecture) for executing dataflow programs was developed to provide an expedient means for executing and testing compiled Id programs. This facilitated final check-out of the compiler and preparation of programs for simulation experiments. The first version of GITA was designed by K. Traub and R. Soley, and implemented in Common Lisp by K. Traub, R. Soley, and D. Morais to run on Symbolics and TI machines [18]. A second version of GITA was developed by D. Culler, G. Papadopoulos, A. Chien, R. Soley, S. Heller, D. Morais, and B. Guharoy which runs in a distributed fashion on the MEF using the circuit switch.

In the abstract model, a configuration of a program is described by an assignment of tokens to arcs in the graph. A node is enabled to execute (or *fire*) when tokens with matching tags are present on its input arcs. When an enabled node fires, it consumes its inputs and generates result tokens on its output arcs. Dataflow programs are determinate, so any enabled node can be fired at any time. Capturing this abstract

model in a real interpreter is surprisingly straight-forward. GITA maintains a queue of unprocessed tokens and a collection of unmatched tokens which have been processed. The main loop dequeues an unprocessed tokens and processes it as follows:

1) If the token requires no partner, the instruction it is destined for is executed and the result tokens are added to the unprocessed queue.

2) If the token requires a partner, the collection of unmatched tokens is searched. If a partner is found, the destination instruction is processed as above.

3) Finally, if no match is found the token is added to the collection of unmatched tokens.

Of course, the name of the game is representation and access to the various data structures, and this has worked out very nicely. GITA currently processes about 2400 dataflow operations per second on a Symbolics 3600 and 1000 on a TI Explorer.

4.3. U-GITA

In addition to executing dataflow programs, GITA provides a basis for modeling certain abstract machines. In particular, it is possible to model the kind of greedy execution one associates with the U-interpreter. Given a configuration with tokens distributed throughout the graph, all enabled activities fire in a single time-step, producing tokens on result arcs. There is no bound place on the number of activities fired and no communication delay. This is realized within GITA by time-stamping the tokens themselves. While executing a dataflow program it is possible to construct *execution profiles* describing the behavior of the program on an ideal machine. Typically, we look at the parallelism (number of operations per time step) and resource requirements, e.g., number of unmatched tokens. It is also possible to model the abstract behavior on a machine with a fixed number of processors.

4.4. MEF-GITA

A collection of Lisp Machines with a high-speed interconnection network is made to act like a collection of (rather slow) dataflow machines by having each execute the GITA interpreter. Tags and I-structure addresses must be meaningful globally, as in a true dataflow machine. Machines spawn work off to other processors and service and request structure operations. All communication is in the form of packets passed across the high-speed network. The resource management issues in this context are essentially as if we had real dataflow machines, except the time-scale is somewhat distorted and the primitives available within each processor are quite powerful. As with the abstract machine, it is possible to construct execution profiles to help understand

the details of program behavior on the machine. The three versions of GITA are one in the same source. Different statistics are collected in the different modes. In fact, we run U-GITA on MEF to construct profiles more rapidly than would be possible with a single Lisp Machine. Work has started on providing a uniform way of feeding statistics collected on these various abstract machines, and the simulator discussed next, into the Illustrate facility.

4.5. SITA: A Simulator for the Tagged-Token Machine

The simulator for the TTDA has been described in previous reports. It provides a detailed picture, essentially register transfer level, of the dataflow machine in operation. The simulator has continued to evolve in response to the needs of the project, and it finally has a catchy name. The changes reflect our growing understanding of the architecture. The simulator was modified to support the bounded loop schema developed by D. Culler [8]. This allows the unfolding of loops to be controlled by a runtime constant in the simulator. Implementation of this feature has allowed us to investigate seriously the effectiveness of proposed loop bounding techniques.

Simulation results have made it clear that current structure allocation schemes fail to take advantage of the pairing of ALU pipelines and I-structure controllers. As a result, little benefit is derived from the local paths between the ALU and the I-structure controller in a PE. In response to this observation, the simulator is being restructured to allow different numbers of I-structure units and ALU pipelines. This does not preclude organizing them in tightly coupled pairs. The restructuring allows us more latitude in experimenting with different machine configurations.

Our computational power was increased considerably last fall by the addition of an IBM 4381. This faster processor allows us to simulate the TTDA at 2.5 times the rate that was previously possible. The simulator on the IBM 4381 now executes approximately 80 dataflow instructions/second.

4.6. Id World

The first version of Id World was completed this past year, integrating tools for editing, compiling, executing, debugging, and analyzing Id programs in an environment similar to what Lisp users enjoy. A new Id Mode within ZMACS allows dataflow programmers to live in the style to which Lisp hackers have become accustomed. From within the editor it is possible to compile meaningful pieces of dataflow programs. The resulting graph is automatically loaded into GITA. On the MEF, the graph is transparently broadcast to all the MEF machines over the circuit switch. When dataflow programs are invoked, GITA is brought into action, so your Lisp Machine (or even 32 of them) look to

you like a dataflow machine. The real *tour de force* is the Id debugger developed by D. Morais. If your dataflow program breaks, you find yourself in the Id debugger, which is like the Lisp machine debugger with some important differences. Instead of a stack trace, you are presented with an invocation *tree*, since after all this is truly a parallel machine. The program structure is physically distributed over the entire MEF, but the debugger makes that entirely transparent.

The other role for Id World is to provide a uniform interface to the three abstract machines. Currently we can record and display profiles from U-GITA or MEF-GITA, but work remains in areas of statistics collection, storage, and presentation.

5. EXPERIMENTS ON MEF

5.1. Dataflow Experiments

The focus of our experiments has been resource management in the TTDA. The resources include token storage, ALU processing cycles, structure storage, and structure memory bandwidth. We have begun to examine how policies for managing these resources effect system performance. The experiments are being conducted on SITA or GITA, according to whichever provides the more powerful tool for highlighting the behavior of interest.

Simulation studies have exposed work distribution and data structure mapping as two crucial issues in constructing a scalable multiprocessor. If we hope to achieve speedup in regions of limited parallelism, work distribution is the critical factor. If we hope to fully utilize the machine in regions of high utilization, then structure contention becomes a very serious problem. Future experiments will further probe these crucial issues.

5.1.1 Token Storage Management

The token storage requirements of tagged-token machines will be quite large as parallelism unfolds and multiple contexts for procedures in the parallel invocation tree are accumulated. In his master's thesis [4], S. Brobst suggests that a hierarchical memory may be an effective way to organize the token storage unit of a tagged-token machine. A token cache can be used to take advantage of temporal locality in the token matching process to yield a high throughput of enabled instructions to the execution unit. For effective cache performance, however, prudent management of program parallelism must be exercised to avoid oversaturation of the token cache. The bounded loop schema developed by D. Culler was shown to have a significant impact in this regard.

5.1.2 Structure Storage Management

In any parallel machine, data structure distribution and contention is an important problem. To this end, A. Chien [7] has examined structure referencing patterns for the SIMPLE code and several smaller programs. Despite the fact that GITA has limited fidelity in timing aspects of the TTDA, these measurements (done with one processor) have given us significant insight into the temporal patterns of structure references. Preliminary indications of "hot spots" have led us to reconsider seriously the benefits of spatial locality in a parallel processor. This has led to a restructuring of SITA, separating the I-structure units from the ALU pipelines and allowing structures to be allocated across structure memory units.

MEF GITA experiments have also given us some insight into the structure storage management problem. We have found that structure contention can be significant. Our initial allocation mechanism places entire structures on a single PE. This, not surprisingly, leads to structure reference contention when a large number of processors are employed. Future experiments will consider the effects of allocating structures consecutive elements are on consecutive processors.

5.1.3 Work Distribution

A number of SITA experiments have focused on the scalability of the architecture. Recent experiments conducted by A. Chien on highly parallel programs show that the architecture tolerates significant network and memory latency with only minimal increases in overall run time. Scalability studies have shown that linear speedups are achievable for programs with large amounts of parallelism. However, careful attention must be paid to work distribution in regimes of low parallelism if the linear speedup is to continue into larger numbers of processors. These results reflect the study of a very limited class of programs.

With each MEF GITA processor pretending to be a dataflow machine, we have strong motivation for developing truly distributed mechanisms for allocating work across the collection of machines. Our approach has been to start simple and look for shortcomings before developing complex mechanisms, such as proposed for the TTDA. Currently, each processor makes independent allocation decisions, without concern for decisions made by other processors. We have experimented with distributing work in a random manner and in ways that take advantage of program structure. So far we have observed good speed-up on most applications, but also certain limitations.

Distributing work on a code-block basis appears to be too coarse. In many cases, there is sufficient instruction level parallelism to keep many processors busy, but not a sufficient number of active code-blocks. This is exacerbated by iniquities in the allocation of work. If most processors have one code block, a few have two, and a few

have none, all may have to wait for the slowest. We plan to investigate finer-grained allocation mechanisms and simple load leveling techniques.

5.1.4 Limitations of MEF GITA

Our initial experiments with MEF GITA have had a dual purpose: (a) to understand the impact of certain resource allocation policies, and (b) understand the bias introduced by MEF GITA as an imperfect execution vehicle. Thus, we have run a collection of applications on a varying number of processors and have begun to cross-check our observations against results obtained on the simulator.

One must be somewhat cautious in drawing conclusions from results obtained on MEF GITA, since there are significant differences between that vehicle and what we expect a real dataflow machine to be. One difference is that in MEF GITA processing structure requests draws away from processing of dataflow instructions (since Lisp Machine cpu cycles devoted to one subtract from those available to the other), whereas in a real machine these would proceed independently. Also, we expect a dataflow machine to be heavily pipelined, whereas MEF GITA is not.

In addition to resource management experiments, MEF GITA has provided a basis for architectural studies. These include (a) analysis of dynamic instruction mixes as compared to conventional machines, (b) token storage requirements of dataflow programs, (c) effects of loop bounding, and (d) impact of reference count storage reclamation.

5.2. DisCoRd: Parallel Graph Reduction on the MEF

Under the guidance of R. Nikhil, P. Lincoln has implemented an emulator for a parallel graph reduction machine on the MEF. It is called DisCoRd, and was part of his bachelor's thesis [17].

Graph reduction is another parallel model of computation for functional programs based directly on the abstract rewriting semantics of functional languages. (Interestingly, recent work by K. Pingali on demand-driven evaluation in dataflow architectures indicates that perhaps one can obtain a unified view of dataflow and graph reduction.)

The system consists of two independent parts: a) a compiler which translates programs in a functional language to a collection of combinator definitions (in ZetaLisp) and a table of strictness information for each combinator,[1] and b) the parallel graph

---

[1]A function is said to be strict in an argument if, whenever the argument is undefined, the function application is also undefined.

reduction machine. We have concentrated mainly on b), leaving open compilation issues in a) for future research.

The parallel graph reduction machine consists of an interconnection of processing elements (PEs). Each PE consists of a reduction engine, a network manager, and a load manager. The reduction engine has a "ready queue" of tasks representing subgraphs to be reduced; for each task it may perform a reduction, or spawn more tasks to evaluate sub-expressions, based on strictness information. The load manager periodically broadcasts the size of the ready queue to neighboring PEs, and maintains an estimate of the sizes of ready queues in neighboring PEs. The network manager is responsible for all communication to other PEs, and uses the load information to decide which PE should receive spawned tasks.

T e DisCoRd system allows the experimenter to specify the number of logical PEs and their logical interconnection topology, and the mapping onto the actual physical resources of the MEF. It starts up Lisp processes representing the logical PEs on the MEF Explorers, establishes the communication links, loads the ZetaLisp combinator code, and starts execution by placing the main program expression graph on one PE's ready queue. DisCoRd is instrumented to measure abstract computation steps and elapsed time.

The DisCoRd system is not yet very stable -- there are several obvious coding improvements to be done, and it uses only Ethernet connections instead of the MEF's circuit switch. We have had time thus far to perform only rudimentary experirnents, reported in P. Lincoln's thesis, which demonstrate the flexibility of the DisCoRd testbed. This summer (1986) R. Nikhil expects to make it more robust and then conduct meaningful experiments to evaluate the feasibility of parallel graph reduction.

6. LANGUAGE RESEARCH FOR THE TAGGED-TOKEN DATAFLOW ARCHITECTURE

Investigation of language-related issues for the Tagged-Token Dataflow Architecture continues under the direction of R. Nikhil, motivated by the belief that such research must be an integral part of any research into parallel architectures.

6.1. Id/83s

The base language we have been using for many years to program the TTDA machines is a functional language called Id. Originally proposed in [3], Id has evolved in a somewhat *ad hoc* and erratic manner over the last few years.

In June/July 1985, R. Nikhil and Arvind used their preparation for the summer course

"6.83s: Functional Programming and Dataflow Architectures" as an opportunity to redesign Id completely, incorporating their latest understanding of issues in data structures and data types. The resulting language is called "Id/83s", and is reported in [20].

6.1.1 I-structures in Id/83s

A fundamental aspect of Id is a data-structuring facility called "I-structures", which attempt to solve what we believe are serious problems in purely functional models of data structures.

In a functional language, data-structures are never "updated". They are created as complete values, and "update" operations create new values. Thus the operation

```
update(a,i,v)
```

where `a,i` and `v` are an array, index and value respectively, produces a *new* array value that is just like `a` except at index `i` where it has value v. In general, the implementation may have to copy the rest of `a`. To minimize this copying, most functional languages provide linked structures (such as lists) built out of "smaller" units such as cons-cells, so that a updated value can *share* as much as possible of the original value. Arrays are then simulated by appropriate operations on linked structures. Unfortunately, for structures that logically behave like arrays, linked structures increase access times (to at least log(n)), make some traversals extremely inefficient, and preclude some parallel accesses.

I-structures were first proposed by Arvind in [2] as a solution to this problem, affording parallel, constant-time access, and avoiding excessive copying. I-structures evolved out of very operational intuitions about data-structures in memory. The key idea was to separate the activity of allocating storage for a data-structure from the activity of filling in its components. Thus an array, when allocated, contains "empty" slots which are later filled by other concurrent operations. To preserve determinacy, only one value is allowed to be written into any slot, and any attempt to read a slot is delayed until it is non-empty.

The semantic implications of I-structures were not so easy to pin down. It was unclear as to what were the appropriate linguistic constructs to manipulate I-structures. For a long time, I-structure constructs in Id clung to a conventional expression-oriented syntax, in keeping with the functional nature of the language.

In the last year, work by K. Pingali has greatly deepened our understanding of I-structure semantics. It is now apparent that with I-structures, Id is no longer in the class of purely functional languages, and can no longer be given classical λ-calculus-based semantics. In particular, Pingali has discovered a close connection between I-

structures and so-called "logic variables" in logic programming languages. He has developed an elegant semantics for functional languages with I-structures, based on solving sets of equations involving so-called "closure" operators.

It was initially difficult for us to realize that we had gone beyond functional languages. I-structure operations do have the flavor of imperatives. However, the constraints on assignment and selection guarantee that programs remain determinate. Id/83s has been designed to reflect this new understanding of I-structures. There are three constructs relevant to I-structures:

- `array(n)` is an expression that allocates an "empty" array of size `n` and returns as value a descriptor to that array. This value is a first-class value.
- `x[i]` is an expression analogous to selecting a component of an array -- it returns the `i`th value of the array `x`, *after* it becomes non-empty.
- `x[i] = v` is a statement analogous to updating an array location -- t stores value `v` in location `i` of array `x`, provided the location is empty. Multiple writes into a location are run-time errors.

Because of the assignment statement, the language is no longer purely expression-oriented --- conditionals, loops, and abstractions can also be statements.

Several programs have been written in Id/83s, including a version of SIMPLE (a moderately large program that is a classic benchmark for evaluating high-performance machines) written by Arvind, giving us some confidence that we have a robust language design.

6.1.2 Types in Id/83s

R. Nikhil has been studying the question of incorporating a data-type system and type-checking into Id/83s. We have become increasingly convinced of the necessity for the compiler to have detailed data-type information about a program. This conviction stems from various sources.

First, in studying the garbage collection problem on the dataflow machine, we have come to realize the necessity of the compiler to determine which arcs in dataflow graphs carry I-structure references and which do not. Without this knowledge, it is necessary to insert conditional garbage collection code at *every* fork point and at every array-select instruction -- this is an unacceptable overhead.

Second, even code that manipulates scalars needs to perform rudimentary type-checking to avoid misintrepretation of scalar data. This is analogous to tag-checking in Lisp implementations, and is also an unacceptable overhead.[2]

[2]On Symbolics machines, ALU operations are done "optimistically" in parallel with tag checking which generates a fault if there is a type-error. Wearing our RISC hats, we believe that this adds unacceptable complexity to the hardware -- most of this checking could and should be done at compile-time.

Third, it is our experience (corroborated by many other researchers) that compile-time type-checking is a powerful debugging tool. Many "silly" errors, such as applying CDR to a number, so typical in Lisp programs and so often discovered after months or years of use of a program, just do not occur in a statically typed system. It is a common folk theorem that when programming in languages like ML or Miranda, most errors are caught by the type-checker.

There are thus two separate but not unrelated questions: a) What data-type system, and b) What type-checking methodology should we adopt for Id/83s?

In answer to question a), we insist that the type system should not seriously constrain the programmer -- one should not have to abandon completely the expressive power of an untyped language.[3] A key to this objective is to adopt a type-system with sufficient *polymorphism*, such as that of CLU or ML. We are looking at the second-order typed λ-calculus as a possibility because we believe that it is a clean and well-understood system that subsumes the type systems of ML and CLU, the only other type systems we know that are practical and meet our needs for expressive power. This type system was proposed independently by Girard in 1971 and Reynolds in 1974.

In answer to question b), we would like the type-checker to perform as much *type-inference* as possible, i.e., to be able to type-check a program without the programmer having to laboriously annotate the code with type declarations. We envisage a facility in Id World where the programmer can point to a sub-expression and ask for its type, and optionally have it automatically inserted. We expect the Id World system to keep track of dependencies between functions, and to redo incrementally the necessary type-checking when a function is edited.

We have been investigating several issues in the use of the Girard-Reynolds type system in Id. It is well known that this type system does not permit full type-inference, i.e., if the programmer were to omit *all* type declarations, the problem of reconstructing those declarations automatically is undecidable. Thus, some annotation is necessary. We are investigating questions such as: How much type declaration is necessary, and is there a simple model for the programmer to decide where to insert such declarations? If essential declarations are omitted, is it acceptable for the type-checker to produce a (weaker) Milner-typing instead? If at times the programmer wishes to bypass the type system (and we believe there will always be such situations) is there a clean way of limiting the scope of this circumvention? The Girard-Reynolds type system is based on a purely functional language (the λ-calculus). Does it extend cleanly to Id, which also has I-structures?

---

[3]The type system of Pascal is an example of one that does *not* meet this goal.

Because of the immediate need for a working language, we did not include a type system as part of Id/83s. However, using intuitions that R. Nikhil has gained from long experience with Milner-type systems, we have designed Id/83s with type-checking in mind. An example of this influence is seen in the separation of lists (homogeneous collections) from tuples (heterogeneous collections). We are confident that we can cleanly augment it with a modern, polymorphic type system.

To answer some of the type-checking questions, S-W. Chen has completed a bachelor's thesis [6] in which he has built an experimental programming system with an incremental Milner type system, along the lines suggested in [21]. Currently, it uses a small functional language typical of many functional languages; we expect to move this system soon into Id World using the new compiler being developed by K. Traub.

6.1.3 Garbage Collection Experiments on MEF

Now that we have started running large experiments on the MEF, the garbage collection problem for the Tagged Token Dataflow machine looms seriously.

Because I-structures are "first-class" objects in Id, they must be allocated dynamically, and reclaimed using some form of garbage collection. We do not think a conventional mark-sweep scheme is suitable for the TTDA, partly because it does not generalize well to the multi-processor situation, and partly because it is difficult, in the TTDA, to identify reliably the "roots" of reachable structures.

We are thus exploring garbage-collection based mainly on reference-counts.[4] The problem still to be solved is this: Because of the non-determinacy in the scheduling of events and in the unpredictability of network transit times, "decrement-refcount" messages and "increment-refcount" messages may arrive at an I-structure in any order. The reference count stored in the structure may thus spuriously drop to zero, and the structure may be reclaimed prematurely.

B. Guharoy is instrumenting the I-structure code in GITA to keep track of reference-count message histories for each structure. We hope to obtain a better idea of how often reference counts go to zero spuriously, and to develop a probabilistic model of how long reference counts stay spuriously at zero. With this understanding, we will be predict with high probability when a reference count is truly zero, and use that as a signal to move that structure to slower storage, reasonably assured that it will never be accessed again.

R. Nikhil has also been looking at program analysis to identify conditions under which

[4]Reference counts cannot be used to reclaim cyclic structures, but we do anticipate cycles to be very common in Id programs.

we can reliably identify I-structures whose extent in time coincides with the lifetime of a code-block. We already have a method of detecting termination of code-blocks; the reclamation of such I-structures may then be triggered by the termination signals of their parent code-blocks, and we can elide their reference-counting instructions altogether.

6.2. Demand-driven Evaluation

Demand-driven evaluation is often described informally as "doing only those computations that are required to produce the output of the program." In his doctoral thesis, K. Pingali defined a functional language L and gave two semantics for it : an operational semantics using dataflow graphs and a denotational semantics using fixpoints on an abstract domain. The informal notion of "minimal computation required to produce the output of an L program" is characterized formally by the least prefix point of the program. He then showed how L programs can be transformed so that a data-driven evaluation of a transformed program performs only these computations. The target language for the transformation was a functional language that is a superset of L. These result can be extended to any functional language without non-sequential functions such as the parallel-or.

Pingali's thesis describes another transformation for achieving the effect of demand-driven evaluation, for which the target language is a functional language in which data structures behave like logical variables in a logic programming language such as Concurrent Prolog. This target language can be looked at as a variation of Id83/s in which I-structure cells can be written into more than once : if a write is attempted in any I-structure cell which already contains some value, the incoming value is unified with the contents of the cell. S. Heller has taken up the task of implementing this transformation. Future work will include:

- defining the notion of demand-driven evaluation for a language with I-structures;
- performing various optimizations to reduce the overhead of the demand code;
- producing graphs which are self-cleaning, i.e., which do not leave tokens behind when the program terminates.

6.3. Databases and Functional Languages

In the last year R. Nikhil has restarted research into databases in functional languages. In [19] he demonstrated the attractiveness and feasibility of the use of functional languages as front-end query languages for existing database systems. The semantic advantages included the availability of an expressive, computationally

complete query language with a rich type system into which conventional data models (such as the relational model) have a simple and natural embedding. He demonstrated an efficient implementation by establishing a correspondence between conventional database operators and stream operators in a graph-reduction execution model for functional languages.

We are now studying the problem of including databases as an integral part of functional languages (rather than relying on an existing, external, conventional database system).

6.3.1 Data Models and Type Structure

In our view, a data model is nothing more than a data type system; a schema for a database is nothing more than a type declaration within that type system; and a database itself is nothing more than an object having the type declared in the schema. With this perspective, it is evident why conventional databases are not used for the hundreds of applications that could use them -- mail systems, operating system tables, text-processor device and font databases, CAD/CAM engineering databases, etc. Conventional data models (even the much-touted relational model) just do not offer the rich type systems that we are accustomed to (and find indispensable) in programming languages. Not only are the type-systems fundamentally inadequate, but they are riddled with *ad hoc* limitations (such as what types may be components of what other types, maximum field sizes expressed in bytes, etc.) that have no semantic justification.

We therefore consider it axiomatic that the fact that an object is to be stored in a database should not be reason for any loss of allowable type structure in that object. Database objects (i.e., those that are stored persistently) should be freely usable with transient objects. It is up to the system implementor to design methods that allow the programmer to store and retrieve *any* object in a type-safe and efficient manner.

6.3.2 Modeling State

At any given point in time, a database is supposed to model an abstracted state of the world. As the state of the world changes with time, the database is expected to keep track of these changed states.

We can ask: How should a database system model the evolving state of the world? Conventional database systems maintain only a "current" state. A change of state in the world is mimicked by erasing some information from, and adding other information to, the database so that once again it represents the "current" state of the world.

We believe that there are many semantic and pragmatic problems with this conventional view. First of all, it seems unlikely that humans model the world this way -- we are capable not only of having an estimate of the current state of the world, but also

of previous states. We may "forget" information, perhaps because it is unnecessary, but we do not consciously erase information. In fact, it can be argued that this capability is necessary to make intelligent decisions about the future.

Second, we believe that erasure of information in databases is motivated mainly from efficiency considerations, and because we are so used to thinking in terms of constructs from imperative programming languages. In many database situations (e.g., banks, personnel, billing) where it is necessary to keep a record of all past states of the database, one usually designs special, *ad hoc* methods to retain information that would normally be erased; these methods are not part of the data model, and are often designed on a per database basis.

When a current state database is shared by many users, it becomes an overly critical resource. Most users do not demand the current state -- a "recent" state is generally adequate. In CAD databases, users often need access to a consistent state of the database for extended periods of time during which they can experiment with various designs. If the contention due to such undemanding users is to be avoided, additional mechanisms must be created to extract snapshots of the database for them.

A central feature of databases is that state transitions, no matter how large, are truly atomic. In a database that maintains only a "current" state, implementing this abstraction is extraordinarily difficult. Further, users who wish only to read the database (no state transition) must contend with those who want to write it (cause a state transition). In a database that models histories of states, these problems are greatly alleviated.

6.3.3 Functional Databases

Our view of databases in functional languages thus emerges. A state of the world is modeled as a persistently stored *environment*, binding names to values (and perhaps types). A query on this state is just an expression evaluated in this environment. A database is a history of states, and an update transaction is a meta-level expression that extends this history by creating a new environment (such as by redefining a function SALARY mapping PERSONs to NUMBERs) based on the existing history. Depending on the application, we may permit only linear history extensions, or we may permit branching extensions -- both have their uses. Extensions occur atomically -- a failed update transaction simply leaves the history unchanged. For linear history extensions, update transactions are serialized to preserve atomic state transitions.

Explicit environment-specification operators in the query and update languages allow queries and updates to depend not only on the most recent state, but on any previous state.

With this philosophical background, we have been a) studying language issues to express this model of state, and b) possible implementation strategies. The major difficulty in a) is to have a view of types that is consistent across histories, i.e., to have meaningful operations that may update a data-type definition from one state to the next.

For implementations it appears that histories of states can be superimposed nicely on graph-reduction models of memory. Each state is associated with its own private graph memory which can share large subgraphs with previous states via pointers to previous graph memories. Thus, a new state initially contains only the portions of the environment that are "updated" with respect to the previous state. Using standard copying garbage-collection techniques, other parts of the new state can be moved up to the new graph memory to improve locality transparently to the user, and old parts of the state history can be cleanly pruned or re-attached dynamically according to available storage capacity.

7. WORK UNDER PROFESSOR DENNIS

7.1. The VIM Project

The VIM project aims to develop an experimental computer system based on principles of data-driven instruction execution and functional programming languages. The project is unique in striving for a system that will serve multiple users with a degree of semantic coherence well beyond what contemporary computer systems are able to offer. It also differs from other efforts to build systems to support functional programming in that the issues of efficient execution of functional programs over a hierarchical memory are addressed and solutions sought. The system uses a base language that is a form of acyclic dataflow graph, and a user language VIMVAL that is an extension and revision of the functional language VAL.

7.2. Accomplishments

In the past five years, substantial progress has been made on VIM. The resolution of successive issues has brought the project to the point where only a few problem areas need to be addressed before a major implementation effort can be mounted.

The project began in 1981 when we realized that our earlier proposals for a general purpose computer system based on dataflow principles could not be adequately evaluated due to the absence of experience in formulating and running functional programs. The nature of programming itself would be so changed by the kind of system we proposed that all past experience would be irrelevant to the detailed design decisions required to define a practical machine.

From the beginning, our plan has been to build an experimental system made of ordinary off-the-shelf components that can evaluate the functional programming style and particular mechanisms for supporting modular programming of a given user community. Once justified by our experience with this unique experimental programming environment, plans could be refined and developed for the kind of powerful and efficient computer systems envisioned in our earlier work.

The first step was the formulation of an operational computer system that encompassed all essential features of the proposed system. This operational model may be viewed as an early simplified specification of the *base language* of our system. Initially the plan of development was to define a series of operational models of increasing detail such that the most refined model was a specification of the VIM implementation -- a set of microcoded routines running on a suitable microprogrammable computer such as a Lisp Machine.

In the academic year 1981-82 the attraction of having a working interpreter for VIMVAL prevailed and an implementation was written by J. Stoy, visiting scientist from Oxford University. Many issues concerning the efficient encoding of instructions and data for the complete language were resolved at that time. In addition, our understanding of the VIMVAL language was refined and improved.

The next step was to develop and evaluate alternate approaches to representing and operating on data structure values stored in the VIM heap. Since the heap is intended to be implemented on a hierarchical memory comprising semiconductor and disk systems, and since efficient concurrent handling of many memory transactions is desired, we proposed using fixed-size *chunks* of memory as the unit of allocation to a data structure. The corresponding creation, augmentation, and access algorithms were designed and specified by B. Guharoy, who also developed suitable mechanisms, based on the reference count technique, for disposing of chunks no longer accessible to computations.

In a major study [14], S. Jagannathan showed how backup and recovery mechanisms may be built into VIM so that users suffer negligible loss of information in the event of a system crash due to any single failure. The technique is based on saving information that permits reconstruction of the results of all function evaluations performed from initiation of a user command to the time of the crash. A small online stable store is proposed to hold the backup data prior to writing it on backup tape.

One significant change in our plans is the switch from a strongly typed language in the spirit of CLU to the use of optional type declarations and a Milner-style type inference technique. The refinement of Milner's theory and an elegant statement of the type-inference algorithm for VIMVAL are given in the prize-winning bachelor's thesis of B. Kuszmaul [16].

Recently, a new operational model for the VIM base language has been formulated by Guharoy and Jagannathan. This work reflects our present thoughts on the elements of the VIM base language. It has been extended to help establish the correctness of the data structure access algorithms and to establish the correctness of the backup and recovery algorithms.

Several new studies have focused on the user programming environment supported by the VIM system. A proposal for the user command language has been drafted by J. Stoy. This proposal raises many questions about the role of environments (which in VIM serve the function of directories in other systems), and the impact of the command language on the VIM type system. These issues are the subject of current doctoral research by E. Waldin.

Other topics for future study include evaluation of approaches to supporting nondeterminate computation using guardians, the application of non-determinacy to data base systems and backtrack programming, and study of language constructs for defining data structure values in general recursive data types.

7.3. Compiling for the Static Dataflow Machine

A compiler that produces efficient dataflow machine code from programs written in the applicative language VAL [1] is crucial to the success of the static architecture in large scale scientific applications. To keep the architecture simple and efficient for computations involving large arrays of numerical data, most of the decisions about resource assignment (allocation of data structures to space in array memory and of dataflow instructions to processing elements) have been entrusted to the compiler.

To accomplish an effective resource allocation, the compiler must transform the program so that its structure is a good match to the processing power and memory space of the target machine. Global program transformations are needed because the several sections of a large program must be capable of operating together in a way that allows full utilization of performance without requiring inordinately large amounts of memory for intermediate results. It is reasonable to attempt such global program transformations because VAL is a functional or applicative language, and therefore interactions among program parts occur only at points that are evident from the syntactic structure of the program -- the impossibility of "side effects" removes the major difficulty that inhibits use of global optimizations in compilers for conventional languages.

A pipelined code mapping scheme as a conceptual basis for such a compiler has been developed in Gao's doctoral research, which will be completed soon. His thesis explores the transformations that can be made to achieve high performance for

numerical programs when executed on a computer based on dataflow principles. We demonstrate how the massive parallelism of array operations in such programs can be effectively exploited by the fine-grain parallelism of static dataflow architecture.

The key is to organize the dataflow machine program graph such that array operations can be effectively pipelined. We introduce a simple value-oriented language to express user programs. Program transformation can be performed on the basis of both the global and local dataflow analysis to generate efficient pipelined dataflow machine code. A pipelined code mapping scheme for transforming array operations in high-level language programs into pipelined dataflow machine programs is described in Gao's doctoral thesis. The machine architecture support for efficient pipelining also is briefly addressed.

Certain results of the thesis research, as well as its application of the pipelined code mapping scheme to some numerical problems, are reported in the publications of the group.

7.4. Simulating Applicative Architectures on the Connection Machine

We have simulated applicative architectures on the Connection Machine. This work was done as a master's thesis by B. Kuszmaul [15].

The Connection Machine (CM) is a highly parallel single instruction multiple data (SIMD) computer, which has been described as "a huge piece of hardware looking for a programming methodology." Applicative languages, on the other hand, can be described as a programming methodology looking for a parallel computing engine.

By simulating architectures that support applicative languages ("applicative architectures"), e.g., dataflow and combinator reduction architectures) on the CM we can achieve the following goals:

- Quickly and easily experiment with the design and implementation of applicative architectures.
- Run large applicative programs efficiently enough to gain useful experience.
- Support programming environments that allow us to do general purpose computation on the CM.

B. Kuszmaul's thesis describes the techniques which we use to simulate applicative architectures on the CM, and the discuss implications for the generalized case of simulating multiple instruction multiple data (MIMD) systems on single instruction multiple data (SIMD) computers.

References

1. Ackerman, W.B. and Dennis, J.B. "VAL--A Value-oriented Algorithmic Language: Preliminary Reference Manual," MIT/LCS/TR-218, MIT Laboratory for Computer Science, Cambridge, MA, June 1978.

2. Arvind and Thomas, R.E. "I-Structures: An Efficient Data Type for Functional Languages," Computation Structures Group Memo 178, MIT Laboratory for Computer Science, Cambridge, MA, October 1981.

3. Arvind, Gostelow, G.P. and Plouffe, W. "An Asynchronous Programming Language and Computing Machine," Department of Computer Science, University of California, Irvine, CA, December 1984.

4. Brobst, S.A. "Token Storage Requirements in a Dataflow Computer," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

5. Bromley, G.F. "Waiting/Matching for Tagged-Token Dataflow Architectures," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

6. Chen, S.W. "A Practical Polymorphic Type-inference Type-checking System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

7. Chien, A.A. "Structure Referencing in the Tagged-Token Dataflow Architecture," Computation Structures Group Memo 268, MIT Laboratory for Computer Science, Cambridge, MA, October 1986.

8. Culler, D.E. "Resource Management for the Tagged-Token Dataflow Architecture," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1985.

9. Gao, G-R. "A Pipeline Code Generation Scheme for Static Data Flow Machine," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1986.

10. Hao, E. "PID Translator User's Manual," UROP Final Report, MIT Laboratory for Computer Science, Cambridge, MA, August 1985.

11. Heller, S.K. and Traub, K.R. "Id Compiler User's Manual," Computation Structures Group Memo 248, MIT Laboratory for Computer Science, Cambridge, MA, May 1985.

12. Heller, S.K. "Directed Cube Networks: A Practical Investigation," Computation Structures Group Memo 253, MIT Laboratory for Computer Science, Cambridge, MA, July 1985.

13. Iannucci, R.A. "Packet Communication Switch for a Multiprocessor Computer Architecture Emulation Facility," Computation Structures Group Memo 220, MIT Laboratory for Computer Science, Cambridge, MA, October 1982.

14. Jagannathan, S. "Guaranteeing Data Security on a Static Data Flow Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, October 1985.

15. Kuszmaul, B.C. "Simulating Applicative Architectures in the Connection Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1985.

16. Kuszmaul, B.C. "Type Checking in VimVal," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

17. Lincoln, P.D. "DisCoRd: Distributed Combinator Reduction," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

18. Morais, D.R. "Id World: An Environment for the Development of Dataflow Programs Written in Id," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

19. Nikhil, R.S. "An Incremental, Strongly-Typed Database Query Language," Ph.D. dissertation, Moore School, University of Pennsylvania, Philadelphia, PA, August 1984.

20. Nikhil, R.S. and Arvind. "Id/83s," Computation Structures Group Memo 249, MIT Laboratory for Computer Science, Cambridge, MA, July 1985.

21. Nikhil, R.S. "Practical Polymorphism," *Proceedings of Functional Languages and Computer Architecture,* LNCS: 201, Springer-Verlag, September 1985.

22. Pingali, K.K. "Demand-driven Evaluation on Dataflow Machines," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, July 1986.

23. Traub, K.R. "A Dataflow Compiler Substrate," Computation Structures Group Memo 261, MIT Laboratory for Computer Science, Cambridge, MA, March 1986.

24. Younis, S.G. "The Clock Distribution System of the Multiprocessor Emulation Facility," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

Publications

1. Arvind and Culler, D.E. "Managing Resources in a Parallel Machine," *Proceedings of IFIP TC-10 Working Conference on Fifth Generation Computer Architecture,* Manchester, England, North-Holland Publishing Company, July 15-18, 1985.

2. Arvind. "Dataflow Architectures," MIT/LCS/TM-294, MIT Laboratory for Computer Science, Cambridge, MA, February 1986. (To appear in *First Annual Review in Computer Science*).

3. Arvind and Iannucci, R.A. "Two Fundamental Issues in Multiprocessing," Computation Structures Group Memo 226-4, MIT Laboratory for Computer Science, Cambridge, MA, January 1986.

4. Brobst, S.A., Malone, T. Grant, K. and Cohen. M. "Toward Intelligent Message Routing Systems," *Proceedings of the 2nd International Symposium on Computer Message Systems*, Boston, MA, September 4-6, 1985.

5. Brobst, S.A. "The Dataflow Model: An Alternative to von Neumann Architectures," *Proceedings of the 5th International Conference in Computer Science*, Santiago, Chile, July 15-17, 1985.

6. Gao, G-R. "A Maximally Pipelined Tridiagonal Linear Equation Solver," *International Journal of Parallel and Distributed Computing*, (1986).

7. Gao, G-R. "A Pipelined Code Mapping Scheme for Tridiagonal Linear Systems," *Proceedings of the 1986 Working Conference on Highly Parallel Computer Architecture*, Nice, France, March 24-26, 1986.

8. Gao, G-R. "Massive Fine-Grain Parallelism in Array Computation: A Dataflow Solution," *Proceedings of Future Directions of Computer Architecture and Software Workshop*, Charleston, VA, May 5-7, 1986.

9. Heller, S.K. "Directed Cube Networks: A Practical Investigation," Computation Structures Group Memo 253, MIT Laboratory for Computer Science, Cambridge, MA, July 1985.

10. Iannucci, R.A. "Dataflow Computer Architecture: An Introduction," Lecture Notes for the International Summer School on Advanced Programming Technologies, Facultad de Informatica, San Sebastian, Spain, September 1985.

11. Nikhil, R.S. "Practical Polymorphism," *Proceedings of Functional Programming Languages and Computer Architecture*, LNCS: 201, Springer-Verlag, Nancy, France, September 1985.

12. Nikhil, R.S. "Functional Databases, Functional Languages," *Proceedings of the Workshop on Persistence and Data Types*, Appin, Scotland, August 1985.

13. Nikhil, R.S. and Arvind. "Id/83s," Computation Structures Group Memo 249, MIT Laboratory for Computer Science, Cambridge, MA, July 1985.

14. Papadopoulos, G.M. "Redundancy Management for Synchronous and Asynchronous Systems, "NATO AGARD Lecture Series 143, NASA Dreyden, CA; Copenhagen, Denmark; Athens, Greece; October 1985.

15. Papadopoulos, G.M. "Design Issues in Data Synchronous Systems," NATO AGARD Lecture Series 143, October 1985.

16. Pingali, K.K. and Arvind. "Efficient Demand-Driven Evaluation (I)," *ACM TOPLAS*, 7, 2, (April 1985), *Corrigendum*: *ACM TOPLAS*, 8, 1, (January 1986).

17. Pingali, K.K. "Efficient Demand-Driven Evaluation (II)," *ACM TOPLAS*, 8, 1, (January 1986).

18. Pingali, K.K. and Kathail, V.K. "An Introduction to the λ-calculus," Computation Structures Group Memo 258, MIT Laboratory for Computer Science, Cambridge, MA, March 1986.

19. Soley, R.M. "Generic Software for the Emulation of Multiprocessor Architectures," MIT/LCS/TR-339, MIT Laboratory for Computer Science, Cambridge, MA, July 1986.

20. Traub, K.R. "A Dataflow Compiler Substrate," Computation Structures Group Memo 261, MIT Laboratory for Compiler Science, Cambridge, MA, August 1986.

Theses Completed

1. Beckerle, M.J. "Logical Structures for Functional Languages," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February 1986.

2. Bromley, G.F. "Waiting/Matching for Tagged-Token Dataflow Architectures," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

3. Brobst, S.A. "Token Storage Requirements in a Dataflow Supercomputer," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

4. Chen, S-W. "A Practical Polymorphic Type-inference Type-checking System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

5. Gao, G-R. "A Pipeline Code Generation Scheme for Static Dataflow Computers," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1986.

6. Gornish, E. "Loop Unfolding for a Static Dataflow Machine," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

7. Hughes, G.W. "A Unified View of Consistency in Fault-Tolerant Computer Design," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, November 1985.

8. Jagannathan, S. "Guaranteeing Data Security in a Static Dataflow Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1985.

9. Kuszmaul, B.C. "Simulating Applicative Architectures on the Connection Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

10. Lincoln, P.D. "DisCoRd: Distributed Combinator Reduction," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

11. Marcovitz, D. "A Comparison of Two Signal System Architectures for a Static Dataflow Machine," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February 1986.

12. Morais, D.R. "Id World: An Environment for the Development of Dataflow Programs Written in Id," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

13. Pingali, K.K. "Demand-Driven Evaluation on Dataflow Machines," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

14. Pinkerton, J.T. "A Method for Translating from a Hierarchical Design System into a Flat Design Structure," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

15. Wanuga, T. "Routing Network Performance in a Static Dataflow Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1985.

16. Younis, S. "The Clock Distribution System of the Multiprocessor Emulation Facility," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

Theses in Progress

1. Chien, A.A. "Congestion Control in Routing Networks," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1986.

2. Chu, T-A. "A Design Methodology for VLSI Self-timed Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.

3. Gao, G-R. "A Pipeline Code Generation Scheme for Static Dataflow Computers," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.

4. Kathail, V.K. "Optimal Evaluators for $\l)$-Calculus Based Functional Languages," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.

5. Kaushik, S. "A Hardware Error Checker for the Packet Switch," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1986.

6. Maa, G. K. "Scalability of the Tagged-Token [illegible] Machine," S.M. thesis, MIT Department of Electrical Engineering [illegible] Computer Science, Cambridge, MA, expected December 1986.

Talks

1. Arvind. "MIT Tagged-Token Dataflow Project," G.E.C., Penta Hotel, London, England, July 15, 1985.
2. Arvind. "Managing Resources in a Parallel Machine," IFIP TC-10 Working Conference on Fifth Generation Computer Architecture, UMIST, Manchester, England, July 18, 1985.
3. Arvind. "Dataflow: A Way of Doing Reduction" and three more lectures on Dataflow and Reduction, First Autumn Workshop on Reduction Machines, Ustica, Italy, September 3-13, 1985.
4. Arvind. "Why Dataflow Architectures?" High Technology Futures, CDC, Riverwood Conference Center, Monticello, MN, September 19, 1985.
5. Arvind. "Dataflow Research in Japan," High Technology Futures, CDC, Riverwood Conference Center, Monticello, MN, September 19, 1985.
6. Arvind. "Why Dataflow Architectures?" Distinguished Lecture Series, Cornell University, Ithaca, NY, October 17, 1985.
7. Arvind. "Why Dataflow Architectures?" Distinguished Lecture Series, Brown University, Providence, RI, October 31, 1985.
8. Arvind. "Demand-Driven Evaluation on Dataflow Machines," Brown University, Providence, RI, October 31, 1985.
9. Arvind. "Why Dataflow Architectures?" Oregon Graduate Center, Portland, OR, November 14, 1985.
10. Arvind. "Why Dataflow Architectures?" Carnegie Mellon University, Pittsburgh, PA, November 22, 1985.
11. Arvind. "Demand-Driven Evaluation on Dataflow Machines," Invited talk, Fifth Conference on Foundations of Software Technology and Theoretical Computer Science, New Delhi, India. December 18, 1985.
12. Arvind. "Why Dataflow Architectures?" Tata Institute of Fundamental Research, Bombay, India, December 20, 1985.
13. Arvind. "Dataflow Architectures," Tata Institute of Fundamental Research, Bombay, India, December 20, 1985.
14. Arvind. "I-Structures," Tata Institute of Fundamental Research, Bombay, India, December 23, 1985.
15. Arvind. "Parallel Machines are Coming," Tata Consulting Services, Bombay, India, December 23, 1985.
16. Arvind. "Why Dataflow Architectures?" Institute Lecture, I.I.T., Kanpur, India, January 1, 1986.
17. Arvind. "Why Dataflow Architectures?" I.I.T., Delhi, India, January 3, 1986.

18. Arvind. "The Dynamic Dataflow Architecture," Advanced Course on New Approaches to the Architecture and the Design of Embedded Systems, E.T.H., Zurich, Switzerland, March 6, 1986.

19. Arvind. "Characteristics of a Processor for a General-purpose Parallel Machine," Workshop on Design and Performance Issues in Parallel Architectures, University of Maryland, College Park, MD. March 17, 1986.

20. Brobst, S.A. "Applying Microcomputers in a Small Organization," AIESEC Spring Regional Conference, Boston, MA, March 8, 1986.

21. Brobst, S.A. "Toward Intelligent Message Routing Systems," 2nd International Symposium on Computer Message Systems, Washington DC, September 6, 1985.

22. Brobst, S.A. "Benchmark Analysis of High Performance MIMD Machines," Harris Corporation, Advanced Technology Division, Melbourne, FL, June 20, 1985.

23. Brobst, S.A. "Performance Evaluation of the Tagged-Token Dataflow Architecture," Workshop on Performance Evaluation of High-Speed Computers, Institute for Computer Sciences and Technology, National Bureau of Standards, Gaithersburg, MD, June 6, 1985.

24. Culler, D.E. "Parallel Processing," Parallel Processing Tutorial, Newport, RI, June 10, 1986.

25. Dennis, J.B. "Dataflow Computing--An Inside View," Cornell University, Ithaca, NY, March 19, 1986.

26. Dennis, J.B. "Dataflow Computing--An Inside View," New York University, New York, NY, March 20, 1986.

27. Dennis, J.B. "Dataflow Computing--An Inside View," Yale University, New Haven, CT, March 21, 1986.

28. Dennis, J.B. "The VIM Project: Experimental Computer Design Using Dataflow Programming Principles, MIT Laboratory for Computer Science, April 23, 1986.

29. Iannucci, R. A. "The MIT Multiprocessor Emulation Facility", MIT Summer Course 6.83s, MIT, August 1985.

30. Iannucci, R.A. Six Lectures on Dataflow Architectures and MEF, International Summer School on Advanced Programming Technologies, Facultad de Informatica, San Sebastian, Spain, September 1985.

31. Iannucci, R.A. "The MIT Multiprocessor Emulation Facility and Dataflow Projects," The University of Manchester, Department of Computer Science, Manchester, England, September 1985.

32. Iannucci, R.A. "The MIT Multiprocessor Emulation Facility and Dataflow Projects," IBM Cambridge Scientific Center, Cambridge, MA, November 1985.

33. Iannucci, R.A. "Dataflow Research at MIT," IBM System Technology Division Headquarters, Endicott, NY, May 1985.

34. Iannucci, R.A. "Dataflow Research at MIT," IBM Data System Division Headquarters, White Plains, NY, September 1985.

35. Iannucci, R.A. "Dataflow Research at MIT," IBM Cambridge Scientific Center, Cambridge, MA, December 1985.

36. Pingali, K.K. "Lazy Evaluation on Dataflow Machines," University of Pennsylvania, Philadelphia, PA, March 20, 1986.

37. Pingali, K.K. "Lazy Evaluation on Dataflow Machines," Microelectronics and Computer Technology Corporation, Austin, TX, March 3, 1986.

38. Pingali, K.K. "Lazy Evaluation on Dataflow Machines," University of Texas, Austin, TX, March 5, 1986.

39. Pingali, K.K. "Lazy Evaluation on Dataflow Machines," Yale University, New Haven, CT, April 1, 1986.

40. Pingali, K.K. "Lazy Evaluation on Dataflow Machines," University of Washington, Seattle, WA, April 8, 1986.

41. Pingali, K.K. "Lazy Evaluation on Dataflow Machines," Stanford University, Stanford, CA, April 10, 1986.

42. Pingali, K.K. "Lazy Evaluation on Dataflow Machines," Cornell University, Ithaca, NY, April 17, 1986.

43. Pingali, K.K. "Lazy Evaluation on Dataflow Machines," University of Illinois, Urbana, IL, May 12, 1986.

44. Nikhil, R.S. "Functional Databases, Functional Languages," Microelectronics Technology Corporation, Austin, TX, June 1985.

45. Nikhil, R.S. "Practical Polymorphism," University of St. Andrews, Scotland, August 1985.

46. Nikhil, R.S. "Functional Databases," Digital Equipment Corporation, Hudson, MA, February 1986.

47. Nikhil, R.S. "Functional Programming Languages," Three lectures in Professor Barbara Liskov's graduate course on programming languages, November 1985.

48. Papadopoulos, G.M. "Redundancy Management for Synchronous and Asynchronous Systems," NATO AGARD Lecture Series 143, NASA Dreyden, California; Copenhagen, Denmark; Athens, Greece; October 1985.

49. Papadopoulos, G.M. "Design Issues in Data Synchronous Systems," NATO AGARD Lecture Series 143, NASA Dreyden, California; Copenhagen, Denmark; Athens, Greece; October 1985.

DISTRIBUTED COMPUTING SYSTEMS

Academic Staff

D. D. Clark, Group Leader
M. Wilkes
J. Saltzer

Research Staff

M. Lambert
J. Romkey

Graduate Students

D. Estrin
D. Feldmeier
J. Gibson
L. Kolodney
P. Ng
J. Sutherland
J. Wolf
L. Zhang

Undergraduate Students

M. Culotta
J. Devine
F. Ehsani
T. Hengeveld
R. Lenoil
J. Lunny
R. Reuss

Support Staff

L. Phelan
G. Staton

1. INTRODUCTION

The Distributed Computing Systems Group has explored a number of aspects of distributed systems, ranging from systems support to semantics of distribution. These various projects are described in the following sections. During the latter part of the year, the activities in the group lessened as attention was focused on the Common System project, described elsewhere in the report.

2. SWIFT

SWIFT is an operating system designed to facilitate efficient network communication. The central feature of the SWIFT operating system is support of a novel modularity technique particularly suited for network protocol software, in which protocol layers may communicate with each other by sub-routine call, rather than by interprocess message. SWIFT executes in single address space for efficiency, with programs written in a high level language to ensure stable and robust operation.

The SWIFT system has been described in greater detail in previous annual reports.

During the current year, work on the SWIFT system was completed. The major activity of the year was a thesis by J. Gibson, which explored the robustness of the single address space in the face of program failure. In the worst case, a program bug in a single address space system can cause arbitrary corruption of any aspect of the environment, because the faulty program can overwrite any region of memory, including those areas critical to overall system operation. SWIFT avoided the worst of these problems by implementing code in CLU, a strongly typed language. The run time checking in CLU prevented such gross errors as referencing outside the bounds of an array or using an arbitrary word of memory as a pointer. These checks are not sufficient to ensure the total health of the system after a faulty program has failed, however. It is necessary to identify and de-allocate the storage associated with that program, and inform any cooperating programs of the failure, so that they, in turn, may make their storage consistent. Gibson's thesis demonstrated that it is possible to build mechanisms which, within a single address base, divide the computations into useful recovery elements, and to dis-entangle and de-allocate one of these elements after a failure. It is thus possible to build an efficient single address space system which is fully as robust and recoverable as the simple multiple address base system such as Unix.

3. RESIDENTIAL CATV-BASED DATA COMMUNICATIONS

As the United States shifts from a manufacturing economy to an information-oriented economy, data processing and computers play an increasingly important role in our work. For people who work with information, it is feasible to work one or more days a week at an office in the home. Working at home has several advantages, including reducing the time and cost of commuting. Work during non-business hours becomes convenient, so that personal schedules may easily be shifted. Work at home also allows one to work during the day without interruptions.

The cost of duplicating office equipment, such as a personal computer, at the office in the home becomes economically competitive with commuting as the price of electronics declines. For work at home to be productive, a worker must have access to the resources available in the office; for example, the use of a mainframe computer or shared data-storage system. The worker at home must not become isolated from the office work environment.

The modern office consists of computers and peripherals connected together with a local area network (LAN). The LAN provides high-speed packet-switched communications over a limited area. To connect the home computer to the office computer system requires a high-speed packet-switching network to the home. A residential cable television (CATV) system is an existing high-bandwidth channel to the home that can provide the basis of such a system.

A recently completed thesis by D. Feldmeier describes a preliminary design for a residential CATV-based high-speed packet-switching network to the home. The thesis considers the theoretical and practical aspects of providing such a service and suggests a possible system design. The system is considered in three parts: transmission from the home to the cable television hub (upstream transmission), transmission from the hub to the home (downstream transmission), and the access scheme.

Upstream transmission is difficult because of the high noise levels on the upstream channel caused by the ingress of short-wave signals and impulse noise. Several noise reduction techniques must be used simultaneously for robust upstream transmission.

The downstream channel has lower noise, but the data signal must be compatible with the CATV system, video signals and television receivers. Vestigial sideband transmission is suggested for total system compatibility.

Existing access schemes, such as those used for local area networks and satellite networks, are unsuitable for a high-speed CATV-based network. Modified versions of two satellite access schemes are suggested as possible solutions. The best techniques for upstream transmission, downstream transmission and access scheme are combined into a single proposed system.

4. LONG ATOMIC TRANSACTIONS

Long atomic computations are computations that may execute for long periods of time but appear to execute serially. In addition, a computation is either committed or aborted. In the latter case, any work performed by the computation is undone and the computation appears to have never executed. In a distributed system, a computation may be delayed from finishing because some nodes in the system are disconnected frequently. The delay can also be caused by unreliable network links or operations that require human interaction. Applications that run on portable computers, such as a personal calendar server, are likely to generate long computations. A computation that sets up an appointment may last for hours or even days because one of the calendars is disconnected from the system.

Because of their lengths, the executions of long computations are more likely to be interleaved or encounter some transient failures. The use of atomicity as the definition of correct behavior relieves the programmers of worries about interferences from concurrent computations or partially executed computations interrupted by failures. However, for many applications, a traditional transaction processing system that uses simple read-write locking does not provide sufficient concurrency. Nor is it desirable to abort a computation whenever a transient failure occurs. The decomposition of a computation into a nested tree of *actions* provides a partial solution: an action can be aborted without undoing the effects of its sibling and ancestor actions. The approach is inadequate as actions near the top of the tree are still vulnerable to transient failures that happen while they are waiting for their children actions to return. A Ph.D. thesis by P. Ng has investigated different solutions to the concurrency and resilience problems in supporting long atomic computations in a distributed system.

We follow the Argus project in modeling the system as a collection of *atomic objects*, each of which provides a set of operations and a *serial specification* specifying the externally visible behavior of the object when there are no concurrency or failures. To guarantee atomicity, each object must ensure that the local knowledge on how computations are ordered are consistent with the rest of the system. The behavior of the object is also constrained to not reflect any incomplete computations (which may be aborted subsequently). We call these constraints *serialization dependencies* and *failure dependencies.* A computation may have to be delayed or restarted when dependencies are created.

Two types of solutions to the concurrency problem are explored. One of them involves adapting the concurrency control algorithms in the system. We have designed algorithms which distinguish between short and long computations. These algorithms eliminate the serialization dependencies created for short computations, and reduce

those created for long computations. The cost of failure dependencies can be reduced with optimistic algorithms. The likelihood of being over-optimistic or cascaded aborts can be minimized with a checkpointing mechanism described below.

Concurrency can also be improved with the use of application semantics. Atomic objects can be implemented with non-atomic objects as long as the external behavior conforms to the serial specification. Application-dependent synchronization and recovery are needed to mask internal concurrency and failures.

We have developed programming paradigms that simplify the writing of such code and help the programmer to convince himself of the correctness of the implementation. We provide a *history* abstraction which captures the local knowledge on what operations have already been invoked at an object, the status of the operations, and the possible ordering constraints on these operations. The history object can be used to determine when dependencies are created. For recovery purposes, a program can follow an *intention list* paradigm in which an operation is inserted into the history when it is invoked (and no dependencies are created), and "merged" with other committed operations when it is committed. If the computation which invoked the operation is aborted, the operation is deleted from the history object. An alternative is an *undo paradigm* in which arbitrary changes can be made in the representation of the atomic object (in addition to inserting into the history), but an application-defined undo operation will be invoked to undo the changes when the computation aborts.

The level of concurrency that can be achieved depends on the semantics of the application as expressed in the serial specification. Interesting classes of functionality-concurrency trade-off can be identified. Concurrency is improved without abandoning atomicity, which has the advantages of being easy to understand and program. This approach compares favorably with other approaches in which atomicity is sacrificed to allow application-defined "interleavings". We can show that any given set of permissible interleavings can be modeled with an equivalent system which consists of atomic objects with relaxed semantics. The atomicity property allows the behavior of the system to be represented succinctly. We are also investigating the classes of applications in which atomicity proves too limited to describe the desirable behavior.

Another aspect of our work on long atomic computations is solutions for the resilience problem. To avoid losing the work of an interrupted long computation, a checkpointing mechanism is needed. We have designed a checkpointing mechanism which allows the programmer to specify enough local state about a computation so that it can be restarted at a checkpoint. Rollbacks due to site crashes, deadlocks, or optimistic concurrency control algorithms can be limited.

In view of the possible long delay to communicate between two sites, we have shown

how the checkpoints within a computation can be coordinated. A program invoking another possibly remote program can execute a checkpoint in anticipation of (or in response to) a long delay in communication. It can also inform its own caller so that its caller can in turn prepare for the delay.

Other than site crashes, a computation may run into communication problems. If the communication network partitions frequently, an operational path linking the sender and receiver may never materialize. Relaying is needed under these circumstances. Because the clients of our relaying service are prepared to handle duplicate messages, we are able to come up with a simple relaying protocol which minimizes the state that needs to be kept by the parties involved. In fact, replication can be used intentionally to increase reliability and minimize delay.

5. RESOURCE MANAGEMENT IN PACKET NETWORKS

Today more and more applications are being brought into packet switching networks, such as packet voice, image data transmission, and teleconferencing. These new applications raise more stringent and diverse performance requirements. They require that the network not only prevent congestion, but also provide a guarantee of various service characteristics. New applications bring another impetus to the research for effective network traffic control and resource management. Their successes, or the further success of packet switching networks, will depend on how well the network can meet these requirements.

L. Zhang continued her work on network resource management and data traffic control. The design of a new network architecture is under way, which will offer clients transmissions services with required delay and throughput performance. The architecture explores a new building block, *flow*, as the basic data transmission unit for network resource management and traffic control. A *flow* is composed of a stream of packets that travel through the same route inside the network, and that require the same transfer performance. A flow differs from packets in datagram networks in that it is treated as one sequence of packets, rather than as independent entities. A flow differs from a virtual connection in VC networks in that it does not concern itself with the connection semantics, such as connection open, close, and state information on data transmissions. Instead, a flow is concerned with the allocation and deallocation of network resources that is required to deliver all data in that flow within the specified performance bounds.

Instead of the conventional window flow control mechanism, the new architecture will explore rate-based data flow control in statistically multiplexed packet switching networks. It will explore a new direction in protocol design and implementations, which

is to coordinate hosts and the network, and to apply the knowledge of data sources' statistical characteristics to network traffic regulation and control.

6. THE NETBLT PROTOCOL

NETBLT (Network Block Transfer) is a transport level protocol intended for the rapid transfer of a large quantity of data between computers. It provides a transfer that is reliable and flow controlled, and is structured to provide maximum throughput over a wide variety of networks.

The protocol works by opening a connection between two "clients" (the "sender" and the "receiver"), transferring the data in a series of large data aggregates called "buffers", and then closing the connection. Because the amount of data to be transferred can be arbitrarily large, the client is not required to provide at once all the data to the protocol module. Instead, the data is provided by the client in buffers. The NETBLT layer transfers each buffer as a sequence of packets, but since each buffer is composed of a large number of packets, the per-buffer interaction between NETBLT and its client is far more efficient than a per-packet interaction would be.

In its simplest form, a NETBLT transfer works as follows. The sending client loads a buffer of data and calls down to the NETBLT layer to transfer it. The NETBLT layer breaks the buffer up into packets and sends these packets across the network in Internet datagrams. The receiving NETBLT layer loads these packets into a matching buffer provided by the receiving client. When the last packet in the buffer has been transmitted, the receiving NETBLT checks to see that all packets in that buffer have arrived. If some packets are missing, the receiving NETBLT requests that they be resent. When the buffer has been completely transmitted, the receiving client is notified by its NETBLT layer. The receiving client disposes of the buffer and provides a new buffer to receive more data. The receiving NETBLT notifies the sender that the buffer arrived, and the sender prepares and sends the next buffer in the same manner. This continues until all buffers have been sent, at which time the sender notifies the receiver that the transmission has been completed. The connection is then closed.

As described above, the NETBLT protocol is "lock-step"; action is halted after a buffer is transmitted, and begins again after confirmation is received from the receiver of data. NETBLT provides for multiple buffering, in which several buffers can be transmitted concurrently. Multiple buffering makes packet flow essentially continuous and can improve performance markedly.

NETBLT has been tested over several different networks with good success. Testing has been conducted using IBM PC/ATs and Symbolics Lisp Machines. We have obtained very high speeds over fast LANs (ethernets and Proteon Ring nets); the PCs

transfer data at a rate of over 1.4 million bits per second, while the Lisp Machines transfer data at about 1.3 million bits per second. Testing of NETBLT over long haul networks has provided interesting results; areas of the protocol, particularly those dealing with timeout strategies, have been extensively overhauled as a result of testing over the 3Mbps Wideband satellite network. Initial tests achieved transfer rates of about 150,000 bits per second. This comparatively low number has been attributed to the relatively low-performance PC ethernet interface; further testing with Lisp Machines should achieve substantially higher transfer rates.

The testing has shown NETBLT to be fairly adept at achieving high transfer rates over different networks. Unfortunately, optimum performance is being hampered by a few problems. While NETBLT provides functionality to negotiate transfer flow-control parameters based on each client machine's resources, it cannot notify intermediate gateways of the flow control parameters. Indeed, one of the problems with testing over the Wideband network has been that the gateways along the transfer path have no knowledge of the client machines' capabilities and are forwarding data too quickly for the client machines to handle.

In order to maximize NETBLT transfer rates, we must come up with a method that allows flow-control negotiation between gateways and client machines as well as between the client machines themselves. The ST protocol (currently being used to carry voice and video packets over the Internet) provides this functionality, although at the expense of forcing a fixed transfer path. Since ST operates in the same layer as IP, it might be possible to use ST to perform flow-control negotiation for NETBLT transfers. Another solution is to invent an IP protocol that performs the flow-control negotiation.

Future research directions for the NETBLT protocol will probably include designing a gateway flow-control negotiation protocol, performance-testing of NETBLT with applications such as video transmission, and continued performance-testing over different networks.

7. PCMAIL: A DISTRIBUTED MAIL SYSTEM FOR PERSONAL COMPUTERS

Pc mail is a distributed mail system that provides mail service to an arbitrary number of users, each of whom has access to one or more personal computers (PCs). The system is divided into two halves. The first consists of a single entity called the "repository". The repository is a storage center for incoming mail. Mail for a Pc mail user can arrive externally from the Internet or internally from other repository users. The repository also maintains a stable copy of each user's mail state (the user's "global mail state"). The repository is therefore typically a computer with a large amount of disk storage.

The second half of Pc mail consists of one or more "clients". Each Pc mail user may have an arbitrary number of clients, which are typically PCs. The clients provide a user with a friendly means of accessing the user's global mail state over a network. In order to make the interaction between the repository and a user's clients more efficient, each client maintains a local copy of its user's global mail state, called the "local mail state". Since clients are PCs, they may not always have access to a network (and therefore to the global mail state in the repository). This means that the local and global mail states may not be identical all the time, making synchronization between local and global mail states necessary.

Clients communicate with the repository via the Distributed Mail System Protocol (DMSP). The repository is therefore a DMSP server in addition to a mail end-site and storage facility. DMSP provides a complete set of mail manipulation operations ("send a message", "delete a message", "print a message", etc.). DMSP also provides special operations to allow easy synchronization between a user's global mail state and his clients' local mail states. Particular attention has been paid to the way in which DMSP operations act on a user's mail state. All DMSP operations are failure-atomic (that is, they are guaranteed either to succeed completely, or fail completely). A client can be abruptly disconnected from the repository without leaving inconsistent or damaged mail states.

Pc mail is a mail system for PCs. Its design has therefore been heavily influenced by several characteristics unique to PCs. First, PCs are relatively inexpensive. This means that people may own more than one PC, perhaps putting one in an office and one at home. Second, PCs are portable. Most PCs can be packed up and moved in the back seat of an automobile, and a few are truly portable -- about the size of a briefcase -- and battery-powered. Finally, PCs are resource-poor. A typical PC has a small amount (typically less than one megabyte) of main memory and little in the way of mass storage (floppy-disk drives that can access perhaps 360 kilobytes of data).

Because PCs are relatively inexpensive and people may own more than one, Pc mail has been designed to allow users multiple access points to their mail state. Each Pc mail user can have several client PCs, each of which can access the user's mail by communicating with the repository over a network. The client PCs all maintain local copies of the user's global mail state, and synchronize the local and global states using DMSP.

It is possible, even likely, that many PCs will only infrequently be connected to a network (and thus be able to communicate with the repository). The Pc mail design therefore allows two modes of communication between repository and client. "Interactive mode" is used when the client PC is always connected to the network. Any

changes to the client's local mail state are immediately also made to the repository's global mail state, and any incoming mail is immediately transmitted from repository to client. "Batch mode" is used by clients that have infrequent access to the repository. Users manipulate the client's local mail state, queueing the changes as "actions". When next connected to the repository, the actions are transmitted, and the client's local mail state is synchronized with the repository's global mail state.

Finally, the Pc mail design minimizes the effect of using a resource-poor PC as a client. Mail messages are split into two parts: a "descriptor" and a "body". The descriptor is a capsule message summary whose length (typically about 100 bytes) is independent of the actual message length. The body is the actual message text, including an RFC-822 standard message header. While the client may not have enough storage to hold a complete set of messages, it can always hold a complete set of descriptors, thus providing the user with at least a summary of his mail state. Message bodies can be pulled over from the repository as client storage becomes available.

A prototype Pc mail system is currently being used by people at LCS and at Project Athena. The repository runs under 4.2 or 4.3 BSD Unix, and client code has been implemented for the IBM PC. Pc mail was discussed at a recent conference on mail systems and personal computers. There was a great deal of interest in the project as a basis for a standard campus-wide mail system. Several enhancements to the basic system were discussed and will probably be added to the Pc mail design and to the prototype implementation. These enhancements include shared mailboxes and mailing list support. The issue of scalability has also been brought up; future work on Pc mail will probably involve writing client code to run on under GNU-EMACS on 4.2/4.3 BSD Unix. A Unix client implementation would allow Pc mail to be used on MicroVax and Sun workstations as well as on Vaxes. This would give us a greater pool of users to test the system (currently relatively few people are in the position of being able to use Pc mail from a networked IBM PC).

8. NETWORK MONITORING

Based on traffic measurements done on the local ring network, R. Jain and S. Routhier proposed a new traffic model. In the new *packet train* model the traffic on the network consists of a number of packet streams between various pairs of nodes on the network. Each node-pair stream consists of a number of trains, with each train consisting of a number of packets (or cars) going in either direction. The inter car gap is large (compared to packet transmission time) and random. The inter train time is even larger. Another observation is that packet arrivals exhibit a *source locality.*

A recently completed thesis by T. Hengeveld measures the network availability of two local area networks: an ethernet and a token ring. His thesis concludes that both networks were very reliable, although both have periods of catastrophic outages. For both networks, the outages were bimodal, with mostly short outages but occasional long outages that accounted for much of the network downtime.

As part of our continuing commitment to excellence in network monitoring, we are beginning a new and exciting project destined to bring insight into the mysteries of higher level protocols communication peculiarities on the token ring network.

A new monitoring project is being undertaken by D. Feldmeier and S. Routhier. The purpose of this monitoring is to gain information about the number and duration of logical connections on the network for applications to flow control and congestion control. Another purpose is to gather more information on packet arrival at hosts on the network, particularly gateways. Unlike previous monitoring which gathered statistics based on low-level, network-specific protocol headers, this project will gather data based on the IP network headers and the port numbers of higher network headers, such as UDP.

A second monitoring project is also being started. This project will be examining the traffic patterns and loading on some of the Laboratory's ethernets. Preliminary results indicate that the ethernet is not highly loaded though the performance is less then we would like. We hope to determine what may be causing this performance loss and to correct it in our next generation of ethernets.

9. ACCESS TO INTER-ORGANIZATION COMPUTER NETWORKS

When two or more distinct organizations interconnect their internal computer networks they form an *Inter-Organization Network (ION)*. IONs support the exchange of CAD/CAM data between manufacturers and subcontractors, software distribution from vendors to users, customer input to suppliers' order-entry systems, and the shared use of expensive computational resources by research laboratories, as examples. A Ph.D. thesis by D. Estrin analyzes the organization implications of using computer networks for inter-organization communication, and the technical implications of interconnecting networks across organization boundaries.

We present a descriptive model of the effects of ION use. IONs change the economics of inter-organization communication. In particular, the speed and incremental cost characteristics support more intense communication, while the capabilities and automatic nature support a greater scope of information and resource sharing across organization boundaries. These enhanced communication patterns in turn allow participants to carry out more activities across their organization boundaries

and with larger numbers of outsiders. At the same time, the ION-supported communication is more penetrating because outsiders access internal resources directly. In addition, when IONs are not universally accessible, communication is segmented between ION and non-ION organizations. These latter two characteristics introduce restrictions which detract from the expansive qualities of IONs. In particular, to compensate for increased penetration organizations may increase formalization of and controls on cross-boundary flows: while segmentation may lead organizations to narrow the range of favored interchange partners to those that are accessible via ION facilities.

We demonstrate the descriptive and predictive value of our general model in the domain of research and development laboratories. This domain provides evidence for our predictions of intensified communication of greater scope and penetration, as well as expanded numbers of cross-boundary activities and interchange partners. We attribute the absence of predicted restrictive behaviors to the absence of resource sharing.

Given our analysis of the organization context in which IONs are used, we demonstrate that such interconnections are satisfied by traditional network design criteria of connectivity and transparency. To the contrary, a primary high-level requirement is access control, and participating organizations must be able to limit connectivity and make network boundaries visible. At the same time, these access control requirements are not satisfied by traditional computer security mechanisms. For example, this investigation of inter-organization networks makes clear that where traditional security mechanisms emphasize information flow, network environments are equally, if not more, concerned with command flow, i.e., invocation of services and applications. We develop a scheme based on non-discretionary controls that allows interconnecting organization to combine gateway, network, and system-level mechanisms to enforce cross-boundary control over invocation and information flow, while minimizing interference with internal operations.

Access control requirements such as these impose new requirements on the underlying interconnection protocols. Just as internetwork access control requirements called for reevaluation of traditional computer security criteria and mechanisms, so cross-boundary connections call for evaluation of traditional approaches to network interconnection. Consequently, we demonstrate the need for alternative interconnection protocols that support loose couplings across administrative boundaries and that accommodate the necessary control mechanisms. Message-based gateways that support non-real-time invocation of services (e.g., file and print servers, financial transactions, VLSI design tools, etc.) are a promising basis for such loose couplings.

The thesis demonstrates the value of our bimodal approach to system design and analysis in which we ask both *how industry and organization contexts shape a new technology*, as well as *how a new technology affects the organization and industry contexts in which is is applied.*

Publications

1. Clark, D.D. "The Structuring Of Systems Using Upcalls," *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, Orcas Island, WA, December 1985.
2. Feldmeier, D.C. "A High-Speed Packet-Switching Network for CATV Systems," *IEEE Transactions on Consumer Electronics*, (August 1985).
3. Feldmeier, D.C. "A CATV-Based High-Speed Packet-Switching Network Design," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, April 1986. Also MIT Laboratory for Computer Science Technical Report, MIT/LCS/TR-304, May 1986.
4. Zhang, L. "Why TCP Timers Don't Work Well," *Proceedings of ACM SIGCOMM '86 Symposium*, August 1986, 397-405.

Theses Completed

1. Estrin, D.L. "Access to Inter-Organization Computer Networks," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1985.
2. Feldmeier, D.C. "A CATV-Based High-Speed Packet-Switching Network Design," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, April 1986.
3. Gibson, J.G. "Computation Management in a Single Address Space System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1986.
4. Hengeveld, T.A. "A Comparison of the Downtime of Two Networks," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.
5. Moel, A. "VMCS 1.01 -- A VM Command Server for the IBM PC-Network," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.
6. Reuss, R. "Computer-Aided Reading" S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.
7. Wolf, J. "An Ethernet Analyzer" SB and MS thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

Talks

1. Clark, D.D. "Why Networks Don't Go Fast," ELECTRO '86, Boston, MA, May 1986.
2. Clark, D.D. "The Future of Communications Research," DARPA, Washington. DC, March 1986.
3. Clark, D.D. "Future Trends in Computer Communications," IBM-ACIS, Boston, MA, March 1986.
4. Clark, D.D. "Upcalls--An Implementation Methodology of Systems," SOSP, Orcas Island, WA, December 1985.
5. Clark, D.D. "A Survey of Current Trends in Networking," IBM University Conference, Santa Clara, CA, November 1985.
6. Clark, D.D. "DARPA Interdomain Addressing" 9th Data Communications Symposium, British Columbia, Canada, September 1985.
7. Clark, D.D. "The Unstated Goals of the DARPA Internet" NSA, Network Security Architecture Working Group, Boston, MA, August 1985.
8. Clark, D.D. "Status Report: DARPA Network Research" Defense Communications Agency, Washington, DC, August 1985.
9. Feldmeier, D.C. "A High-Speed Packet-Switching Network for CATV Systems," IEEE International Conference on Consumer Electronics, Chicago, IL, June 1985.
10. Ng, P. "Long Atomic Computations," University of Rochester, Rochester NY, February 1986; University of Illinois at Urbana-Champaign, Urbana Il, March 1986; University of California, Los Angeles CA, March 1986; University of Southern California, Los Angeles CA, March 1986; Carnegie-Mellon University, Pittsburgh PA, April 1986.
11. Zhang, L. "A Congestion Control Fix for the ARPA Internet," Internet Engineering Task Force meeting, Washington DC, April 1986.

INFORMATION MECHANICS

Research Staff

T. Toffoli, Group Leader

G. Vichniac

Graduate Students

N. Margolis

P. Tamayo

Undergraduate Student

D. Petty

Support Staff

T. Cloney

D. Zaig

Visitor

E. Goles

1. INTRODUCTION

The Information Mechanics Group has been undergoing some reorganization, following the departure of E. Fredkin as a Group Leader. The production and documentation of CAM-6 and the writing of a book on cellular automata machines (see below) have created a work overload; added to this is the need to publish our results in a timely manner in highly competitive areas such as fluid dynamics modeling. As a result, we have been forced to queue the publication of other results and an adequate treatment of a number of new ideas.

2. CELLULAR AUTOMATA MACHINES

The modeling philosophy embodied in the Cellular Automata Machine (CAM) architecture has been extensively documented in a book, Cellular Automata Machines -- A New Environment for Modeling [5], which will be published in the Fall by the MIT Press. This book, conceived as a standalone work, will do double duty as a complement to the documentation for CAM-6. A trickle of CAM-6's is finally being produced; we are struggling to get the manufacturer to ramp up production.

N. Margolis has single-handedly completed the immense task of writing a usable and reliable software package for driving in real time up to eight CAM-6 modules through the IBM PC; this package provides a high level, structured environment for interactive experimentation with cellular automata models. To him goes the credit also for a number of hardware revisions that make the machine more powerful and conceptually simpler to use, and for insuring that the final product indeed performs according to specifications.

A. Califano has been writing the CAM-6 user manual and developing primitives for statistical experiments with the machine. T. Cloney has interfaced CAM-5 with a CADR, and has been running it as a Lisp Machine co-processor for experiences using cell arrays much larger than those directly provided by CAM-5. We have initial results on reflection, refraction, and diffraction of sound and solitonic waves through non-homogeneous media realized as lattic gases.

3. THE CAM-7 MULTIPROCESSOR

We have completed the essential design of a cellular-automaton multiprocessor, called CAM-7, which will be especially useful for fine-grained models of physical systems. In its basic configuration (a module that should cost about $400,000 for a limited production run) CAM-7 will process 10 giga-events/sec (where an *event* is the updating of a 2-bit cell as a function of up to 16 neighboring bits), and handle a cube of 512x512x512 sites. The architecture is modularly expandable.

We intend to go ahead with construction as soon as we are free of other engagements and the funding prospects are a little clearer.

4. FLUID DYNAMICS MODELING

The idea of modeling fluids by means of cellular automata ("lattice gas" models), introduced by Pomeau in 1973 and independently rediscovered by us a few years later, has recently caught the attention of the scientific community. One of the stimuli for the more recent developments has been the availability of cellular automata machines -- which are ideally suited for simulating lattice gases. For these applications, the new architecture offers a performance factor of 1000 to 10,000 over the architecture of ordinary scientific computers; as a result, what had been introduced merely as conceptual models are becoming effective computational tools. Some of our work in this area has been published in *Phys. Rev. Letters [3]* and mentioned (in a badly digested form) in *Nature*.

5. COMBINATORIAL DYNAMICS

We have obtained results of a truly novel kind: one concerning the reduction of variational principles to combinatorial dynamics, and the other concerning the reduction of special relativity to combinatorial dynamics. These and other results in information mechanics have been pushed on the stack, waiting for their turn to be written up.

We have also continued work on quantum computation [4], and on the construction of energy-like invariants in reversible cellular automata [1].

6. CA '86

T. Toffoli has been organizing, with C. Bennett and S. Wolfram, a conference, CELLULAR AUTOMATA '86, to be held at MIT on June 16-19, 1986. This conference will bring in about 150 participants, of which about 50 will give contributions in the form of lectures, posters, presentations, demos, etc. Full-length papers will go through refereeing and will be published as conference proceedings.

7. NEURAL NETWORKS

E. Goles and G. Vichniac have constructed Lyapunov functions for parallel neural networks [2]. The field of neural networks has enjoyed a renaissance since 1982 after J. J. Hopfield has expressed complex networks dynamics as an optimization process, where the local minima of Lyapunov functions can be used as content-addressable

memories. Our work extends Hopfield's result to the exciting case of parallel network dynamics, and connects neural networks to cellular automata.

8. STATPHYS-16

G. Vichniac has been active in the organization of STATPHYS-16, a large physics conference that meets every three years. He has insured a good representation of the MIT Cellular Automata Machine (CAM), physics of computation, cellular automata theory, and neural networks.

References

1. Goles, E. and Vichniac, G. "Invariants in Automata Networks," submitted to *Journal of Physics A.*

2. Goles, E. and Vichniac, G. "Lyapunov Functions for Neural Networks with Parallel Iterations," to appear in *Neural Networks for Computing 1986,* J. S. Denker (ed.), American Institute of Physics, 1986, 165-181.

3. Margolus, N., Toffoli, T. and Vichniac, G. "Cellular Automata Supercomputers for Fluid Dynamics Modeling," *Physical Review Letters,* 56, (1986), 1694-1696.

4. Margolus, N. "Quantum Computation," *Proceedings Conference on New Ideas and Techniques in Quantum Measurement Theory,* Annals NY Academy Science, 1986.

5. Toffoli, T. and Margolus, N. *Cellular Automata Machines: A New Environment for Modeling,* Cambridge, MA, MIT Press. 1987.

Publications

1. Brower, R. C., Giles, R. and Vichniac, G. "Serial and Parallel Computation of Ising Spins," to be submitted to *Physica D.*

2. Cloney, T., Goles, E. and Vichniac, G. "The '3*x*+1' Problem: a Quasi-Cellular Automaton," to be submitted to *Proceedings of the Cellular Automata '86 Conference,* C. H. Bennett, T. Toffoli and S. Wolfram (eds.). (See MIT/LCS/TM-317 for extended abstract.)

3. Giraud, B. G. and Vichniac, G. "Effective Forces and Rigorous Variational Principles," *Physical Review* A, 32 (1985) 72-80.

4. Goles, E. and Vichniac, G. "Invariants in Automata Networks," submitted to *Journal of Physics A.*

5. Goles, E. and Vichniac, G. "Lyapunov Functions for Neural Networks with Parallel Iterations," to appear in *Neural Networks for Computing 1986,* J. S. Denker (ed.), American Institute of Physics, 1986, 165-181.

6. Hartman, H., Klein, W., Tamayo, P. and Vichniac, G. "Statistical Mechanics of Inhomogeneous Cellular Automata," to be submitted to *Proceedings of the Cellular Automata Conference,* C. H. Bennett, T. Toffoli and S. Wolfram (eds.). (See MIT/LCS/TM-317 for extended abstract.)

7. Hartman, H. and Vichniac, G. "Inhomogeneous Cellular Automata (INCA)," in *Disordered Systems and Biological Organization* (Les Houches 1985 Winter School, Proceedings), E. Bienenstock, F. Fogelman, and G. Weisbuch (eds.), NATO ASI, Series F: Computer and System Sciences, 20, Springer-Verlag, 1986.

8. Margolus, N., Toffoli, T. and Vichniac, G. "Cellular Automata Supercomputers for Fluid Dynamics Modeling," *Physical Review Letters*, 56, (1986), 1694-1696.

9. Margolus, N. "Quantum Computation," *Proceedings Conference on New Ideas and Techniques in Quantum Measurement Theory,* Annals NY Academy Science, 1986.

10. Myczkowski, J. and Vichniac, G. "Critical Behavior in Cellular Automata Models of Growth," to be submitted to *Proceedings of the Cellular Automata '86 Conference*, C. H. Bennett, T. Toffoli and S. Wolfram (eds).

11. Toffoli, T. and Margolus, N. *Cellular Automata Machines: A New Environment for Modeling*, Cambridge, MA, MIT Press. 1987.

12. Toffoli T. and Margolus, N.H. "The CAM-7 Multiprocessor: A Cellular Automata Machine," MIT/LCS/TM-289, MIT Laboratory for Computer Science, Cambridge, MA, 1985.

13. Vichniac, G. "Taking the Computer Seriously in Teaching Science (An Introduction to Cellular Automata)," in *Proceedings of the UNESCO Conference on Microcomputers and Science Education,* G. Marx (ed.), Budapest, Hungary, 1985.

14. Vichniac, G. "Cellular Automata Models of Disorder and Organization," in *Disordered Systems and Biological Organization* (Les Houches 1985 Winter School, Proceedings), E. Bienenstock, F. Fogelman and G. Weisbuch (eds.), NATOASI, Series F: Computer and System Sciences, 20, Springer-Verlag, 1986.

15. Vichniac, G., Hartman, H. and Tamayo, P. "Annealed and Quenched Inhomogeneous Cellular Automata (INCA)," to appear in *Journal of Statistical Physics.*

16. Vichniac, G. "Noncrystalline Cellular Automata," to be submitted to *Proceedings of the Cellular Automata '86 Conference*, C. H. Bennett, T. Toffoli and S. Wolfram (eds.).

17. Vichniac, G. "Boolean Calculus on Cellular Automata," to be submitted to *Proceedings of the Cellular Automata '86 Conference*, C. H. Bennett, T. Toffoli and S. Wolfram (eds.). (See MIT/LCS/TM-317 for extended abstracts).

PROGRAMMING METHODOLOGY

Academic Staff

B. H. Liskov, Group Leader
W. E. Weihl

Research Staff

C. D. Chambers
P. R. Johnson
S. E. Perl
R. W. Scheifler

Graduate Students

G. Chan
J. C. Chung
M. S. Day
K. J. Goldman
D. J. Hwang
E. K. Kolodner
R. Ladin
G. T. Leavens
B. M. Oki
S. E. Perl
M. E. Vermeulen
E. F. Walker

Undergraduate Students

C. D. Chambers
G. Greeley
C. L. Mullendore
H. Woo

Support Staff

A. L. Rubin

Visitor

R. Leidhammar

1. SUMMARY

During the past year we have continued our work on the design and implementation of Argus, and we have also succeeded in bringing up an implementation of CLU on the M68000. In addition, we have continued our study of general issues in distributed systems, including work on concurrency control techniques [40], and on techniques for converting single-site programs into servers suitable for use over a network [6]. In the sections that follow we discuss some of this work in more detail. Section 1.1 discusses our work on Argus over the past year. Section 2 describes an application written in Argus and some preliminary conclusions about the expressive power of Argus. Section 3 describes a technique developed for specifying the behavior of distributed systems. Finally, Section 4 describes work on the availability provided by a number of replication techniques.

1.1. Argus

Argus is a programming language and system that supports the construction and execution of distributed programs that run on a number of computers connected by a network. It provides novel linguistic mechanisms, in particular, guardians, which allow a distributed program to be modularized in an effective way, and atomic actions, which allow computations in a distributed environment to behave properly in spite of concurrency and failures. Argus is described in [24][25].

During the past year we have continued working on both the design and implementation of Argus. In the area of design, we have made a number of rather small changes to the language that enhance its usefulness, and have documented these changes and the current status of Argus in a revised reference manual [23]. One important change is the addition of closures to Argus. We have also studied the *name clash* problem. This problem arises in polymorphic interfaces that require their type parameters to have certain operations. For example, consider a parameterized *sort* routine that works on array[t] provided t has an operation named "<" that compares two elements of type t to see if one is less than the other. Now consider a type S that has an operation that does the less-than test but that has a different name than "<". If *sort* requires that the operation be named "<", it would not be possible to pass an array[S] to *sort*, and the usefulness of *sort* would be more limited than necessary. We have worked out a general scheme for avoiding this problem in Argus. The scheme works for types like S where the operation exists but has the wrong name; it also allows a type without the needed operation to be extended by providing a procedure. Note that the name clash problem also arises in object-oriented languages such as SmallTalk.

We have continued to work on the Argus implementation, and have completed a

prototype that runs on Vaxes under Berkeley Unix 4.2. A major accomplishment has been the incorporation of stable storage into the implementation. The design of our stable storage system is described in [30].

Another major accomplishment is the design and implementation of the X-Window system. This system was initially developed primarily for use in the Argus system, especially in the debugger. It has since been adopted by Project Athena as the primary window system for their workstations, and by DEC for some of their products.

Much of the implementation effort has been focused on debugging. An interactive debugger has been implemented, permitting users to control the execution of processes and to examine the values of variables and objects while programs are running. This approach to debugging is described in [36]. The debugger's user interface uses the X-Window System to present each guardian and process in a separate window. This interface seems much easier to use than an earlier one that required the user to manage many processes through the same window.

2. EVALUATION OF ARGUS

Our experience in writing application programs has led to some tentative conclusions about Argus. In general, the experience has been quite positive: it is easy to get a prototype of a distributed application up and running quickly. In addition, our work on applications written in Argus, such as the attempt by R. Leidhammar to write a telephone switching system, has given us insight into the expressive power of Argus. There are some areas where we have noticed difficulties; these are discussed below.

In this section we summarize our experience implementing CES, a distributed Collaborative Editing System, which was developed by R. Seliger and implemented in Argus. We extract from CES a set of representative requirements for distributed applications, and evaluate the support provided by Argus for meeting those requirements. The material in this section is abstracted from [13].

As its name implies, CES serves several purposes. First, it provides long-term storage of structured documents, such as books and articles. Second, it provides an editing facility that is based on document structure. Third, it permits several authors to collaborate in writing a document. Collaboration can take many forms. For example, different authors could edit the same section of a document at different times, each completing a draft before the next begins editing. Alternatively, co-authors could edit different sections of a document simultaneously.

CES was implemented as a distributed application, rather than on a single centralized machine, partly to explore the needs of and methods for building distributed

applications, and partly because the requirement to support collaboration demands a distributed implementation in today's network environment. For example, co-authors might work at geographically distributed locations. CES could not be implemented as a centralized system using "remote login" facilities to provide access for remotely located users without violating the application's autonomy and availability requirements, which are discussed below.

CES exhibits a number of requirements common to many distributed applications:

1) *Distribution.* CES must support clients using geographically distributed computers connected by a network. Clients should have to deal with the distributed nature of the system as little as possible; i.e., the system should be *location transparent* as much as is possible.

2) *Sharing.* Clients should be able to share documents and sections of documents. This is essential to support collaboration.

3) *Autonomy.* Documents, or sections of documents, should belong to individual clients. A client should be able to control the sections of documents that he owns; he should be responsible for supplying storage to hold them, and only them, and should be able to prevent others from accessing them.

4) *Reliability.* The system should be *resilient* to crashes and other failures (e.g., network problems), in that it should be very unlikely to lose data because of a failure. In addition, the system should be *available* in the presence of failures: even when one or perhaps several machines have crashed, clients should be able to access information stored in the system.

5) *Reconfiguration.* The system should be easy to reconfigure, so that sites can be added and deleted as the hardware configuration changes.

6) *Concurrency.* The system should permit users of the system to edit documents at the same time. In doing so, the delays encountered by one user because of another user's activities should be kept as small as possible. (The latter is necessary to meet the performance requirement below.)

7) *Consistency.* The data stored in the system should remain consistent even in the presence of concurrency and failures. Notice that this may require some limitations on concurrency.

8) *Performance.* The system should provide reasonably fast response to users. Notice that the distributed nature of the system can both help and hurt performance. The ability to run several computations concurrently on different machines may improve performance. At the same time, the need to send messages over the network to accomplish a task may hurt performance.

9) *High-Quality User Interface.* The semantics of the system as seen by users should be reasonably simple; for example, the distributed nature of

the system, and the presence of concurrency and failures, should not cause unusual or surprising behavior to occur. In addition, the editor's interface should permit easy access to a document's structure, or outline, in addition to the text in the document.

Obviously, some of these requirements conflict with each other. For example, the desire for sharing can conflict with the need for autonomy. In addition, the requirements should not be taken as absolutes. For example, regardless of how the system is implemented, it will always be possible for a sufficient number of failures to cause data to be lost, or to make data inaccessible temporarily.

As noted above, CES was implemented in Argus. Argus was designed to support the construction of reliable distributed applications, and thus contains features targeted at many of the requirements listed above. For example, Argus provides *guardians*, which are the units of distribution. Guardians can be created and destroyed dynamically, and thus support reconfiguration. *Stable storage* can be used in guardians to store data that needs to survive crashes. Argus also permits concurrent and shared access to data stored in a guardian. *Atomic actions* can be used to ensure consistency in the presence of concurrency and failures.

The features provided by Argus made it quite easy to meet many of the requirements listed above. For example, the distribution, sharing, and autonomy requirements were easily met by using guardians on each client's computer to store the parts of documents owned by that client. The resilience requirement was met by a combination of stable storage and atomic actions, while the availability requirement was met by implementing a standard replication algorithm [12] to enhance availability in the presence of failures. Similarly, the consistency requirement was met by using atomic actions, while the concurrency requirements were met in part by keeping atomic actions relatively short, and in part by implementing user-defined atomic types [38][39].

Argus also proved to be an excellent development and debugging environment. The system manages network communications and installation of guardians at remote sites, and provides a powerful set of distributed debugging tools. In addition, atomic actions proved to be very convenient for controlling concurrency and handling failures such as site crashes.

In several areas, however, we found the requirements for CES difficult or awkward to meet. In general, the problems arose because of combinations of requirements, rather than individual requirements. For example, the concurrency and performance requirements dictate a high level of concurrency in the processing of concurrent users' commands. At the same time, the consistency requirement dictates certain limits on concurrency. Atomic actions were used to ensure consistency; to achieve a high level of concurrency, user-defined atomic types [38][39] were implemented. However, as has

been noted by Weihl [38][37], the mechanisms provided by Argus for building user-defined atomic types are limited in their expressive power, primarily because no user code runs when an action commits or aborts. We encountered similar limitations.

The requirement for a high-quality user interface led us to implement CES using a high-resolution bit-mapped display, using separate areas of the screen to display a document's structure and particular parts of a document. The display can be viewed as a kind of cache for the data stored in the system. Obviously, a user would like to see characters appear on the display as they are typed. To meet the performance requirement, however, it is necessary to include several editing changes in a single atomic action. If such an action aborts, the display must be updated to be consistent with the data stored in the system. Keeping the display consistent with the internal data as atomic actions commit and abort, however, turned out to be awkward and somewhat inefficient.

The resilience requirement was met by using stable storage, and the concurrency requirements were met in part by keeping atomic actions short. Stable storage is updated whenever a top-level atomic action commits. (Argus provides *nested* actions, not just single-level actions.) If actions are short, stable storage can be updated frequently. To meet the performance requirements, the amount of data to be written to stable storage when a given action commits must be kept relatively small. Argus provides mechanisms for controlling the granularity at which data is written to stable storage. However, we encountered modularity problems arising from the existence in Argus of two encapsulation mechanisms for data: a conventional data abstraction mechanism for local data within a guardian, the *cluster* (borrowed from CLU [19]), and guardians themselves, which provide operations that permit access by remote users. Part of the reason for this distinction in Argus is historical: Argus is based on CLU, and it was convenient to retain clusters for local data. However, as discussed in more detail in [13], there are also technical reasons why this distinction may be important. In fact, many other systems have a similar distinction: one mechanism is used to build local data, while another is used to build remotely accessible objects (e.g., see [5][1][35]). One of our conclusions is that some of the functionality available in building remotely accessible objects would be useful in building local data as well.

A final problem involved the desire for a simple user-visible semantics. In certain cases, a user could type some characters in a part of a document and then display that part of the document on a different part of the screen, and the characters just typed would not appear until after several seconds. This behavior is at best surprising. The source of the problem involves the manner in which the Argus system propagates information about the commits and aborts of multi-site atomic actions from site to site. In fact, the current semantics of Argus appear to make it impossible to avoid this kind of

surprising behavior. Subtle changes to the language semantics would make it relatively straightforward to solve this problem. However, these changes appear to require a more complex and less efficient implementation. Current research is directed at understanding whether this problem can be solved efficiently.

The design of CES represents one way of resolving the conflicts between the various requirements; different designs would lead to different detailed requirements, and might result in a system that would be easier to implement. However, we feel that the problems we encountered in implementing CES are quite general, and that they illustrate basic difficulties with certain aspects of the design of Argus. It is important to emphasize that our overall experience in using Argus to implement CES was quite positive: a reasonably efficient implementation of a complex distributed application was built in a relatively short time. The problems we encountered indicate a need for further research to deepen our understanding of techniques for implementing distributed applications.

3. SPECIFICATIONS OF DISTRIBUTED PROGRAMS

In [22], Liskov and Weihl propose a method for specifying distributed programs. Such programs often have a number of performance requirements that make them difficult to specify. First, they are typically concurrent and may require a high degree of concurrency. In addition, they may need to be highly reliable (i.e., unlikely to lose information entrusted to them) or highly available (i.e., needed information is likely to be accessible), and they may need to have fast response time. These requirements often have an effect on the functional behavior of a program, forcing designers to change their initial expectations. In this section, we sketch how the method from [22] can be used to give user-oriented specifications of the functional behavior of programs with such requirements.

Like user-oriented specifications of sequential programs, specifications of distributed programs should be expressed in user-oriented terms and should be free of implementation detail. We show how to give such high-level specifications. We claim that our approach need not lead to the loss of any information that is of use to the users of a system, and that it is possible to give a high-level specification that accurately describes the relevant aspects of a system's behavior. We support this claim in [22] by giving specifications of a number of programs in the literature, one of those examples is summarized below.

We view a distributed system as an abstract object that can be used by calling various operations. Thus, the system is an instance of an abstract data type [26][21][14]. The specification of a system describes all relevant constraints on its observable behavior;

this includes the behavior of the operations called by users and, if the system is active, internal operations that are run by the system itself. The user of a system could be a program or, if the system is interactive, a person. A system may have concurrent users, and these users may invoke its operations in parallel.

In our specifications, each operation is viewed as an atomic action. Atomic actions have been widely studied in recent years as a way of organizing programs that must cope with concurrency and failures (e.g., see [9][3][32][28][27]). Atomic actions have two important properties: serializability and totality. *Serializability* means that the concurrent execution of a group of actions is equivalent to some sequential execution of the same actions. *Totality* means that each action is all-or-nothing: either it executes successfully to completion (in which case we say that it *commits*), or it fails and has no effect on the state of the system (in which case we say that it *aborts*).

Viewing each operation as atomic greatly simplifies specifications. It means that operation invocations, even those executing in parallel, never appear to overlap; instead two such calls appear to occur in some sequential order. Furthermore, if for some reason it is impossible to complete a call, it aborts. This means that the specification is simpler and easier to understand for both users and implementors of the system.

In effect, what we get from atomicity is the ability for a user to view the program as sequential even when it is not. The state of the program when some operation is invoked is viewed as the result of the executions of all earlier committed calls, where these executions are performed in their serialization order. In other words, in the specification we can assume that operations are executed sequentially rather than concurrently. Furthermore, we need only consider operations that commit; effects of aborted actions can be ignored.

Note that a specification that views each operation as an atomic action does not constrain the implementation to execute each operation atomically. Several authors (e.g., see [4][10][11][17] [34]) have claimed that atomicity is incompatible with satisfying various requirements such as high availability or high concurrency. However, we can give specifications of systems with such requirements and with atomic operations. The authors mentioned above are right that atomicity at the level of the implementation is sometimes incompatible with performance requirements like availability and concurrency; however, this does not preclude using atomicity in the specification. An analysis of the concurrency and availability permitted by atomicity can be found in [38][16].

We propose a structure for specifications that distinguishes expected and desirable effects from undesirable ones. (A similar approach was proposed in[18].) We believe that this distinction is an important one for both users and implementors of a system,

and that it makes the specifications easier to understand. Our specifications are informal, in the sense that we do not give a precise mathematical meaning for them. An important open problem is to provide a precise meaning for these kinds of specifications.

In the remainder of this section, we look at a simple example, taken from [10], and show how atomicity can be used to write a high-level specification that avoids implementation details. In this example, the goal is to achieve high availability. We also summarize the other examples from [22]. Our approach is essentially bottom-up: we start with a description of the implementation of a system, and then discuss how to specify that system in user-oriented terms. We take this approach to illustrate the expressive power of our specification technique. In designing and implementing a system, we would suggest a more top-down approach.

Fischer and Michael present an efficient algorithm for maintaining a replicated distributed dictionary, with *insert*, *delete*, and *list* operations [10]. The dictionary is modeled as a set, with *insert* adding an element to the set, *delete* deleting an element from the set, and *list* returning the current members of the set.

The dictionary is to be implemented on a distributed system, consisting of a collection of nodes connected by a network; both the nodes and the network may be unreliable. The goal is to make the system highly available, in the sense that any operational node should be able to perform any of the three operations at any time, regardless of the status of the network or of other nodes.

The implementation works by processing each operation at a single node. Each node has a copy of the dictionary. If an *insert* or *delete* operation is executed at a node, only that node's copy of the dictionary is updated. Messages are sent between nodes periodically to propagate information about updates. A *list* operation executed at a node returns exactly those elements known by the node to have been inserted and not known to have been deleted. (A node knows about a previous operation if either the operation was executed at the node, or the node has received a message from another node that knew about the operation at the time the message was sent.) Because one node may not know about some operations performed at other nodes, users are required to ensure that any given element is inserted at most once, that an element is deleted only if it has been inserted at some point in the past, and that the node at which a deletion occurs knows about the prior insertion.

Notice that this replicated dictionary does not meet the usual specification of a dictionary. In particular, a *list* operation does not return exactly those elements that have been inserted and not yet deleted. Instead, it returns only those elements that are known at the node at which it is executed. Thus, a specification of this system must reflect the replicated nature of the implementation in some way.

Fischer and Michael s specification, however, contains implementation details that are unlikely to be of interest to a user of the system. For example, a user cannot see or control the internal communication, yet the specification of the dictionary depends heavily on this communication. These implementation details also introduce the possibility of over-specification, placing unnecessary and possibly undesirable restrictions on implementations.

insert = **proc** (x: element)
    requires $x \notin$ Members.
    effects Adds *x* to Members.

delete = **proc** (x: element)
    requires $x \in$ Members.
    effects Adds *x* to ExMembers.

list = **proc** ( ) **returns** (sequence[element])
    effects Returns a subset of Members.

Figure 7-1: Specifications of the dictionary's operations.

In Figure 7-1 we present an alternative specification of the dictionary that hides these details. In the specification, we treat the dictionary as a logically centralized resource; its distributed implementation is irrelevant. We model the state of the dictionary as a pair of sets, Members and ExMembers. Members contains all elements that have been inserted, while ExMembers contains all elements that have been deleted. Each of the three operations described by the specification is viewed as an atomic action. The operations preserve the following invariant on the state:

ExMembers $\subseteq$ Members

The figure illustrates the form we use for specifications. The specification of each operation consists of a *header* followed by a number of clauses. The header describes the types of the arguments and results. The optional *requires clause* describes the assumptions that the operation makes about its arguments and the state of the system when it is called. For example, the requires clauses for the *insert* and *delete* operations define two assumptions needed for Fischer and Michael's algorithm. Finally, the *effects clause* describes the behavior of the operation when the assumptions stated in the requires clause are satisfied; if the requires clause is not satisfied when the operation is called, then the specification places no constraints on the behavior of the operation.

The figure also illustrates the way we approach specifications. We begin by introducing an abstract model [26] for the system state; this model is used in defining each of the operations. The operations are then specified operationally, i.e., they modify or read the state.

As with any abstract data type, the operations of the dictionary can be classified as *constructors*, which modify the state, and *observers*, which read the state. (Sometimes an operation is both a constructor and an observer.) For the dictionary, *insert* and *delete* are constructors, and *list* is an observer. In all our specifications, the constructors have an exact effect on the state, e.g., *insert* really adds the new element to Members. However, users can only observe the state by calling the observers. Since these typically do not allow the entire state to be observed, e.g., list only returns a subset of the state, users cannot necessarily see the effects of the constructors.

The key idea in these specifications is the use of nondeterminism to model the delay between the execution of the constructors and the propagation of their effects on the state to later operations. Thus, the specification of *list* is nondeterministic. Using nondeterminism in this way enables us to hide both the existence of multiple copies of the dictionary at different nodes, and the communication among nodes.

The specification in the figure does not match the specification given earlier in one respect: the requires clause of *delete* does not state the constraint that the node at which the deletion is executed must know about the insertion of the element to be deleted. To state this requirement, we would need to expose the existence of nodes in the specification. It would not be difficult to add the nodes to the specification. (We would model the multiple copies in the state. In addition, the operations would all have as an extra argument the node at which the call is to run.) We chose not to do this because our specification suppresses information that is not of interest to users, and is thus more user-oriented.

When it is impossible to give an accurate description of a system without exposing information that is not relevant to users of the system, we should consider whether different decisions should be made, so that the specification need not expose such details. One virtue of specifications is that they serve as a warning flag by bringing such questions to our attention. They help the designers to make an informed decision, where the impact on the user is well-understood.

Of course, a specification is of interest only if it can be implemented efficiently. For example, can the specification in the figure be implemented as efficiently as the algorithm of Fischer and Michael? We have done such an efficient implementation for a system very similar to the dictionary. Our system is part of an orphan detection algorithm [20]; its specification is essentially the one given in the figure. Its implementation must store information about deletes explicitly, where Fischer and Michael need not do this, but we can use garbage collection to discard this information as soon as all nodes know about the delete.

The specification given above describes the behavior of the dictionary in high-level,

user-oriented terms, but does not capture some important aspects of its behavior. In particular, it does not indicate what a *list* operation is likely to return. What is needed is an approach that permits us to distinguish the normal, expected behavior (e.g., that *list* returns exactly those elements that have been inserted but not deleted) from other kinds of behavior that, although still possible, are less likely and less desirable.

We suggest the following simple approach. divide the effects clause of a specification into two parts, the *normal* and the *abnormal* effects. The *normal* effects describe what the user can ordinarily expect to observe, e.g., when all nodes are up and information is being propagated among nodes fast enough. In effect, it describes an ideal system in which all modifications are observable immediately. The *abnormal* effects describe unlikely events, e.g., what happens when there is a problem like a network partition that prevents messages from being sent from one node to another. It describes situations where the implementation only approximates the ideal system.

In Figure 7-2, we show a revised specification for the *list* operation of the replicated dictionary; the specifications of the other operations remain unchanged. Notice that the normal behavior of *list* is to return exactly those elements that have been inserted and not deleted. It is still possible for it to return some elements that have been deleted, or not to return some elements that have been inserted; this behavior is described as the abnormal effects. Notice also that the abnormal effects simply repeats the normal effects but in a "fuzzier" way. All our specifications are like this; the abnormal effects happen when the normal effects can only be approximated.

list = **proc** ( ) **returns** (sequence[element])
    normal effects Returns Members – ExMembers.
    abnormal effects Returns a subset of Members.

Figure 7-2: Revised specification of *list*.

A specification using this notation should be interpreted as defining a nondeterministic choice between the normal and abnormal effects. The normal effects are intended to be much more likely to occur than the abnormal effects, but a user of the operation must be prepared for either. A reader would mostly pay attention to the normal effects, since this is what would happen most of the time. He would have to be aware of the abnormal effects, but would not expect them to happen very often.

Our specification, however, does not describe the relative likelihoods of the various outcomes of an operation. To really understand how to interpret "normality", a reader needs to know the relative likelihoods of the various outcomes of an operation. For example, it should be likely that an element that was inserted a long time ago and has not been deleted will be returned by a call of the *list* operation, and similarly if an element was deleted a long time ago it should be unlikely that it will be returned.

Intuitively, the normal effects should be likely to happen provided the call happens long enough after all other calls whose effects it is supposed to observe. We do not know how likelihood should be expressed formally; we discuss this issue further below.

In [22], we also give specifications for two other systems: a banking system taken from [11], in which the main issue is concurrency, and a real and very complicated system, Grapevine [4], in which the major concerns are availability and fast response time. In both cases we are able to apply the specification method sketched above to give high-level, user-oriented specifications of the systems.

In our specifications, we used a format in which normal and abnormal effects were specified separately. As discussed earlier, the idea is that a reader will mostly pay attention to the normal effects, since this is what should happen most of the time. He will have to be aware of the abnormal effects, but should not expect them to happen very often. To really understand how to interpret "normality," however, the reader needs to know the relative likelihoods of the various outcomes of an operation. It might be useful to associate probabilities with the different outcomes of an operation, or to rank them in order of preference. It is not clear, however, how to give a precise mathematical meaning to such specifications. This is an area where more work is needed.

Another way of specifying systems like these is to use the "eventually" operator of temporal logic [29] to require that the results of an update will eventually be visible to later operations. This gives a result similar to what we obtain using nondeterminism, since nothing is said about *when* the results will become visible. However, there are several problems with using "eventually." The first is that it may be too strong, since if enough failures occur at inopportune times it is possible (though perhaps extremely unlikely) that something might *never* happen. This should not mean, however, that the system has failed to meet its specification. For example, in Grapevine, a message might never be delivered, and no trouble message sent either, but this does not mean that the system is broken. A specification that required all messages to be delivered eventually would not be satisfied by Grapevine, or by any realistic implementation.

The second problem with "eventually" is that it does not say enough. Simply requiring that something happen eventually does not tell a user how long it can be expected to take to happen. This is also a problem with our specifications, since they do not indicate how long one should wait to be reasonably certain that the normal effects will happen. This, too, is an area that deserves further study.

Another problem is that a specification that requires something to happen eventually does not indicate which results are more desirable than others. We believe that the distinction between normal and abnormal effects made in our specifications is important

for both users and implementors of a system. For example, such specifications could serve as a basis for comparing implementations. One implementation might be considered better than another if the abnormal effects occur less frequently in it, or if the normal effects happen more quickly. Again, this is an area where more work is needed.

In our specifications, we view each operation as an atomic action. As mentioned earlier, this greatly simplifies specifications, yet does not overly constrain implementations. Notice also that this does not preclude multi-step behavior. For example, in our specification of Grapevine, we introduce an internal operation that actually delivers mail; the operation called by users to send mail does not deliver the mail immediately, but instead queues it for later delivery. In effect, Grapevine is a "mail spooler" just like a spooler for an I/O device. In any such system, there must be a source of activity that is independent of operations called by a user to carry out the work that is done later. In the specification of Grapevine in [22], we modeled this activity by the internal operations *deliver* and *reap*. Like all operations, these extra operations are atomic.

We claim that our specifications accurately describe the aspects of each system's behavior that are relevant to the system's users. For example, in the Fischer-Michael example, knowing that the replicas send messages to one another does not help the user understand how quickly a given addition or deletion will propagate. Similarly, in Grapevine, knowing that servers communicate by sending messages does not help the user understand how soon mail will arrive. Therefore, our specifications, in which the delay in information propagation is modeled by nondeterminism, are as informative to the user as this extra detail. (The extra detail is clearly of interest to a system implementor or maintainer, but our specifications are not intended for them.)

The advantage of specifications is that they make it clear exactly what interface a program provides. Specifications provide valuable information not only to users of a system, but to designers. Users, of course, need to know what behavior they can expect to see. For example, a user of Grapevine needs to know all the situations that might happen, even if some are unlikely to occur (e.g., the problems with adding and deleting names, registries, and servers). Similarly, designers need to decide if they have made the right decisions. Given the specifications, they can decide whether the interface is what is wanted. Different designers might make different decisions; the important point is that they need a clear understanding of a system's interface in order to decide if the trade-offs made during design are justified.

Our approach here has been primarily bottom-up: we started with a description of the implementation of each system, and then described how to specify that implementation in user-oriented terms. Program design, however, should be driven primarily by the

user's requirements, and thus should take more of a top-down approach. For example, one might start with an relatively strong specification (e.g., without any nondeterminism), and gradually weaken it by introducing nondeterminism and abnormal effects as the implementation constraints become better understood. At each stage, the specification could be used to help evaluate decisions, since it makes evident the effects on the users.

4. LIMITATIONS ON AVAILABILITY IN THE PRESENCE OF PARTITIONS

In designing fault-tolerant distributed database systems, a frequent objective is to make the system highly available in spite of component failures. We measure availability as the fraction of transactions presented to the system that complete. One technique to increase data availability is replicating the data at various sites in the network. In [7], we examine several replicated-data management protocols that maintain database consistency and attempt to make replicated data highly available in the presence of network partitions. (Partitions are failures that divide a system into two or more components between which communication is impossible.)

The protocols we examine in this paper maintain one-copy serializability [2][31], and are of the *on-line* kind, that is, those that are required to make irrevocable commit/abort decisions at the time the transaction is processed. We do not consider other classes of protocols, such as *off-line* protocols that can defer commit/abort decisions until the partitions are rejoined; protocols that abandon one-copy serializability as the correctness criterion; and protocols that use type-specific information. Davidson's optimistic protocol [8] is an example of an off-line protocol. The partition-tolerant distributed databases project at the Computer Corporation of America [33] is an example of a system that abandons one-copy serializability to achieve higher availability. Herlihy [15] deals with replication methods for abstract data types.

The main objective of replicated-data management protocols is achieving availability while maintaining data consistency. No protocol whose correctness criterion is one-copy serializability can do better than a bound we have determined, under the assumption that the pattern of data accesses by transactions obeys a certain uniformity assumption that we explain. We believe this assumption is a reasonable one if we know nothing in particular about the transaction distributions; there might be some particular distributions for which specialized protocols achieve greater availability. Furthermore, this assumption permits us to do the analysis.

In the context of a simple model we have developed, we analyze the level of availability achieved by several replicated-data management protocols proposed in the

literature. The protocols we look at use different rules to increase data availability during a partition. Given the authors' informal discussion of availability achievable by these protocols, it is difficult to determine how one protocol compares against the others. We provide a uniform basis for comparison. In addition, we show that several of the protocols achieve the upper bound for availability, so the bound is tight.

Our analysis shows that there is a severe limitation on the availability that can be achieved during a partition. Because of this limitation, networks should be designed to minimize the probability that partitions will occur. In the rest of this section, we present the bound on availability. An analysis of existing protocols, showing the extent to which they achieve the maximum possible availability, can be found in [7].

4.1. Assumptions and Definitions

The context of our work is a *distributed database system* in which the data is fully replicated. This system consists of a collection of n sites, numbered $1,...,n$. We have chosen this special case of a distributed database system because it simplifies our analysis. *Transactions* can operate on data items by reading or updating. We assume no blind updates, where a blind update is one that updates a data item without first reading it. The set of data items updated by a transaction is called its *write set*.

A *partition* occurs in a system when two functioning sites are unable to communicate for a significant interval of time. A maximal set of sites that can communicate with one another is called a *partition group*, following Davidson [8]. We make the simplifying assumption that a partitioned network consists of only two partition groups, called a *majority partition* and a *minority partition*. This simplification does not change our conclusions because we believe that this kind of partition is rare and that more extensive partitioning is even rarer; we analyze the most common case. We define L_j as the *load* on the system for a site j based on the fraction of transactions that run there. That is, L_j is the number of transactions initiated at site j divided by the total number of transactions.

Then a set S of sites is a *majority* if and only if $\Sigma_{s \in S} L_s > 1/2$. We use this somewhat nonstandard definition of majority because it ensures the desirable property that during a partition more than one-half of the work submitted to the system is submitted to the majority partition.

In this paper, we are interested in availability. It is a measure of the amount of work that can be done by a system. We define availability as follows: the availability achieved during an execution is the number of transactions successfully completed divided by the total number of transactions presented to the system. We do not study other aspects of performance, such as the relative expense of read/write operations or the cost of rejoining partitions.

In order to quantify our observations concerning availability, we are interested in the following parameters.

- t = total number of transactions presented during the partition
- u_{maj} = fraction of t that are update transactions and are in the majority partition
- u_{min} = fraction of t that are update transactions and are in the minority partition
- r_{maj} = fraction of t that are read-only transactions and are in the majority partition
- r_{min} = fraction of t that are read-only transactions and are in the minority partition

4.2. Analysis

We now show that no on-line replicated-data management protocol that maintains one-copy serializability can achieve a level of availability that is better than $u_{maj} + r_{maj} + r_{min}$.

Our proof depends on an assumption regarding system workload, which we call the *uniformity assumption*. The uniformity is described formally in [7]. One informal characterization, which is sufficient for the uniformity assumption to be satisfied, is that the transaction mix be the same at each site. For example, suppose 10% of the update transactions run at site 1; then our assumption says that of those transactions that update a set of data items, 10% of them run at site 1.

We find it convenient to formulate the following correctness property, which in Theorem 1 we show is a necessary condition for maintaining one-copy serializability.

Correctness property. For all data items d, d is not updated on both the minority and majority sides of a partition.

Theorem 1. Any replicated-data management protocol that preserves one-copy serializability satisfies the correctness property.

The following lemma is central to our proof of the upper bound bound on the availability achievable by a replicated-data management protocol that maintains one-copy serializability.

Lemma 2. Let a be any replicated-data management protocol that satisfies the correctness property and that operates in a system in which the uniformity assumption holds. Any execution e of protocol a during a partition has availability at most $u_{maj} + r_{maj} + r_{min}$.

The following theorem provides an upper bound on availability.

Theorem 3. In a system where the uniformity assumption holds, no replicated-data management protocol that maintains one-copy serializability can achieve availability greater than $u_{maj} + r_{maj} + r_{min}$.

Proof. Assume that there exists an protocol *a* that has availability greater than $u_{maj} + r_{maj} + r_{min}$. By Lemma 2, protocol *a* violates the correctness property. By Theorem 1, protocol *a* does not preserve one-copy serializability. Contradiction. □

Our analysis shows that there is a severe limitation on the availability that can be achieved during a partition. Because of this limitation, networks should be designed to minimize the probability that partitions will occur. We believe that it is technically feasible to make network partitions highly unlikely by building sufficiently reliable networks using a combination of the following techniques: high connectivity; software security, such as preventing malicious application programs from corrupting the operating system or gateway; and physical security, such as keeping people away from the machines.

But even though the likelihood of a network partition may be small given such a sufficiently reliable network, software should still preserve one-copy serializability in the presence of partitions. A system should not fail in a catastrophic way. When choosing a replicated-data management protocol from among several, the criteria one uses need not necessarily be availability alone; other factors, such as ease of implementation and cost of repairing a database when partitions rejoin, should be considered as well.

References

1. Allchin, J. E. "An Architecture for Reliable Decentralized Systems," GIT-ICS-83/23, Georgia Institute of Technology, September 1983.

2. Bernstein, P. A. and Goodman, N. "Multiversion Concurrency Control -- Theory and Algorithms," *ACM Transactions on Database Systems*, 8, 4, (December 1983), 465-483.

3. Bernstein, P. A. and Goodman, N. "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, 2, 13, (June 1981), 185-221.

4. Birrell, A., Levin, R., Needham, R. and Schroeder, M. "Grapevine: An Exercise in Distributed Computing," *Communications of the ACM*, 25, 4, (April 1982), 260-274.

5. Black, A. "Supporting Distributed Applications: Experience with Eden," *Proceedings of the Tenth Symposium on Operating Systems Principles*, December 1985, 181-193.

6. Chung, J. "Converting Single-host Application Programs into Network Services," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1986.

7. Coan, B., Oki, B. and Kolodner, E. "Limitations on Database Availability When Networks Partition," *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing*, August 1986.

8. Davidson, S. B. Optimism and Consistency in Partitioned Distributed Database Systems," *ACM Transactions on Database Systems*, 9, 3, (September 1984), 456-481.

9. Eswaran, K. P., Gray, J. N., Lorie, R. A. and Traiger, I. L. "The Notions of Consistency and Predicate Locks in a Database System," *Communications of the ACM*, 19, 11, (November 1976), 624-633.

10. Fischer, M. J. and Michael, A. "Sacrificing Serializability to Attain High Availability of Data in an Unreliable Network," *Proceedings of the Symposium on Principles of Database Systems*, March 1982.

11. Garcia-Molina, H. "Using Semantic Knowledge for Transaction Processing in a Distributed Database," *ACM Transactions on Database Systems*, 2, 8, (June 1983), 186-213.

12. Gifford, D.K. "Weighted Voting for Replicated Data," *Proceedings of the Seventh Symposium on Operating Systems Principles*, December 1979, 150-162.

13. Greif, I., Seliger, R. and Weihl, W. "A Case Study of CES: A Distributed Collaborative Editing System Implemented in Argus," Programming Methodology Group Memo 55, MIT Laboratory for Computer Science, Cambridge, MA, April 1987.

14. Guttag, J., Horowitz, E. and Musser, D. "Abstract Data Types and Software Validation," *Communications of the ACM*, 12, 21, (December 1978), 1048-1064.

15. Herlihy, M. "A Quorum-consensus Replication Method for Abstract Data Types," *ACM Transactions on Computer Systems*, 4, 1, (February 1986), 32-53.

16. Herlihy, M. P. "Replication Methods for Abstract Data Types," MIT Laboratory for Computer Science, Cambridge, MA, 1984.

17. Lamport, L. "Towards a Theory of Correctness for Multi-user Data Base Systems," Report CA-7610-0712, Massachusetts Computer Associates, Wakefield, MA, October 1976.

18. Lampson, B. "Atomic Transactions," in *Distributed Systems: Architecture and Implementation*, LNCS: 105, Goos and Hartmanis (eds.), Berlin, Springer-Verlag, 1981, 246-265.

19. Liskov, et al., "CLU Reference Manual, LNCS: 114, Berlin, Springer-Verlag, Goos and Hartmanis (eds.), 1981.

20. Liskov, B. "Overview of the Argus Language and System," Programming Methodology Group Memo 40, MIT Laboratory for Computer Science, Cambridge, MA, February 1984.

21. Liskov, B., et al. "Abstraction Mechanisms in CLU," *Communications of the ACM*, 8, 20, (August 1977), 564-576.

22. Liskov, B. and Weihl, W. "Specifications of Distributed Programs," *Distributed Computing*, 1, (1986), 102-118.

23. Liskov, B., et al. "Argus Reference Manual," Programming Methodology Group Memo 54, MIT Laboratory for Computer Science, Cambridge, MA, March 1987.

24. Liskov, B. and Scheifler, R. W. "Guardians and Actions: Linguistic Support for Robust, Distributed Programs," *ACM Transactions on Programming Languages and Systems*, 5, 3, (July 1983), 381-404.

25. Liskov, B. "Overview of the Argus Language and System," Programming Methodology Group Memo 40 , MIT Laboratory for Computer Science, Cambridge, MA, February 1984.

26. Liskov, B. and Zilles, S. N. "Programming With Abstract Data Types," *Proceedings of the ACM SIGPLAN Conference on Very High Level Languages*, April 1974, 50-59.

27. Liskov, B. and Scheifler, R. "Guardians and Actions: Linguistic Support for Robust, Distributed Programs," *ACM Transactions on Programming Languages and Systems*, 3, 5, (July 1983), 381-404.

28. Moss, J.E.B. "Nested Transactions: An Approach to Reliable Distributed Computing," MIT/LCS/TR-260, MIT Laboratory for Computer Science, Cambridge, MA, 1981.

29. Owicki, S. and Lamport, L. "Proving Liveness Properties of Concurrent Programs," *ACM Transactions on Programming Languages and Systems*, 3, 4, (July 1982), 455-495.

30. Oki, B., Liskov, B. and Scheifler, R. "Reliable Object Storage to Support Atomic Actions," *Proceedings of the ACM Tenth Symposium on Operating Systems Principles*, 1985.

31. Papadimitriou, C. H. "The Serializability of Concurrent Database Updates," *Journal of the ACM*, 26, 4, (October 1979), 631-653.

32. Reed, D.P. "Naming and Synchronization in a Decentralized Computer System," MIT/LCS/TR-205, MIT Laboratory for Computer Science, Cambridge, MA, 1978.

33. Sarin, S. "Robust Application Design in Highly Available Distributed Databases," *Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems*, 1986, 87-94.

34. Schwarz, P. and Spector, A. "Synchronizing Shared Abstract Types," *ACM Transactions on Computer Systems*, 3, 2, (August 1984).

35. Spector, A. Z., et al. "Support for Distributed Transactions in the TABS Prototype," CMU-CS-84-132, Carnegie Mellon University, July 1984.

36. Vermeulen, M. "Debugging with Remote Procedure Calls," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1986.

37. Weihl, W. "Linguistic Support for Atomic Data Types," *Proceedings of the Workshop on Persistence and Data Types*, Scotland, August 1985.

38. Weihl, W. E. "Specification and Implementation of Atomic Data Types," MIT/LCS/TR-314, Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, MA, March 1984.

39. Weihl, W. and Liskov, B. "Implementation of Resilient, Atomic Data Types," *ACM Transaction on Programming Languages and Systems*, 7, 2, (April 1985).

40. Weihl, W. E. "Distributed Version Management for Read-only Actions," *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, Minaki, Canada, August 1985.

Publications

1. Coan, B., Oki, B. and Kolodner, E. "Limitations on Database Availability When Networks Partition," Programming Methodology Group Memo 49, MIT Laboratory for Computer Science, Cambridge, MA, May 1986. Also to be published in the *Proceedings of the 5th ACM Symposium on the Principles of Distributed Computing.*

2. Greif, I., Seliger, R. and Weihl, W. "Atomic Data Abstractions in a Distributed Collaborative Editing System," *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Programming Languages*, January 1986.

3. Liskov, B. and Guttag, J. *Abstraction and Specification in Program Development*, Cambridge, MA, MIT Press, and New York: McGraw Hill, 1986.

4. Liskov, B., Herlihy, M. and Gilbert, L. "Limitations of Synchronous Communication with Static Process Structure in Languages for Distributed Computing," *Proceedings of the 13th ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, January 1986

5. Liskov, B. and Ladin, R. "Highly-Available Distributed Services and Fault-Tolerant Distributed Garbage Collection," Programming Methodology Group Memo 48, MIT Laboratory for Computer Science, Cambridge, MA, May 1986. Also to be published in the *Proceedings of the 5th ACM Symposium on the Principles of Distributed Computing.*

6. Liskov, B. and Weihl, W. "Specifications of Distributed Programs," *Journal of Distributed Computing 1*, 2 (April 1986), Springer-Verlag. Also Programming Methodology Group Memo 46, MIT Laboratory for Computer Science, Cambridge, MA, September 1985.

7. Oki, B., Liskov, B. and Scheifler, R. "Reliable Object Storage to Support Atomic Actions," *Proceedings of the ACM Tenth Symposium on Operating Systems Principles*, December 1985. Also Programming Methodology Group Memo 45, MIT Laboratory for Computer Science, Cambridge, MA, September 1985.

8. Weihl, W. E. "Distributed Version Management for Read-Only Actions," *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, Minaki, Canada, August 1985. Also Programming Methodology Group Memo 47, MIT Laboratory for Computer Science, Cambridge, MA, May 1986.

9. Weihl, W. "Linguistic Support for Atomic Data Types," Workshop on Persistence and Data Types in Database Programming Languages, Appin, Scotland, August 1985.

Theses Completed

1. Chambers, C. D. "A New Method for Implementing Atomic Types," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1986.

2. Chung, J. "Converting Single-host Application Programs into Network Services," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

3. Greeley, G. "An Ada Implementation for Fault Detection Using Multiple Processors," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

4. Mullendore, C. L. "The Issues of Implementing CLU Using the Language C," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1986.

5. Woo, H. "An Abstract Object Browser for the Argus Environment," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

Theses in Progress

1. Bela, L. "Winsrv Support for Common ASCII Display Terminals," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.

2. Chan, G. "Generalized Transaction Management," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1987.

3. Day, M. "Replication and Reconfiguration in a Distributed Mail Depository," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.

4. Kolodner, E. "Recovery from Garbage Collected Virtual Memory," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.

5. Perl, S. "Distributed Commit Protocols for Nested Transactions," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1987.

6. Vermeulen, M. "Debugging with Remote Procedure Calls," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.

Invited Talks

1. Day, M. "Mail Protocols and Highly-Available Mailboxes," Workshop, Stanford University, Stanford, CA, May 1986.

2. Liskov, B. "Argus: Linguistic Support for Distributed Computing," Brown Boveri, Baden, Switzerland, August 1985.

3. Liskov, B. "Software Engineering," Panel Discussion, Wang Institute of Graduate Studies, Tyngsboro, MA October 11, 1985.

4. Liskov, B. "Implementation of Resilient, Atomic Data Types," Stanford University, Stanford, CA, January 29, 1986.

5. Liskov, B. "Specifications of Distributed Programs," Stanford University, Stanford, CA, January 30, 1986.

6. Liskov, B. "Overview of Argus"; "2-Phase Commit Protocol," Asilomar Workshop on Fault Tolerant Distributed Computing, Monterey, CA, March 1986.

7. Liskov, B. "Specifications of Distributed Programs," IBM Research Center, San Jose, CA, March 1986.

8. Oki, B. "Reliable Object Storage to Support Atomic Actions," 10th ACM Symposium on Operating Systems Principles, Orcas Island, East Sound, WA, December 2, 1985.

9. Weihl, W. "Distributed Version Management for Read-Only Actions," Fourth Annual ACM Symposium on Principles of Distributed Computing, Minaki, Ontario, August 5, 1985.

10. Weihl, W. "The Uses of Atomicity in Distributed Systems," University of Cambridge Computer Laboratory, Cambridge, England, August 26, 1985.

11. Weihl, W. "Linguistic Support for Atomic Data Types," Workshop on Persistence and Data Types in Database Programming Languages, Scotland, August 30, 1985.

12. Weihl, W. "Linguistic Support for Atomic Data Types," University of Massachusetts, Amherst, November 1, 1985.

13. Weihl, W. "The Uses of Atomicity in Distributed Systems," Carnegie-Mellon University, Pittsburgh, PA, November 11, 1985.

14. Weihl, W. "Specifications of Distributed Programs," University of Toronto, Canada, April 17, 1986.

PROGRAMMING SYSTEMS RESEARCH

Academic Staff

D. Gifford, Group Leader

Research Staff

S. Berlin

Graduate Students

R. Baldwin
C. Chiang
M. C. Chua
N. Glasser
J. Lucassen
S. Martin
J. O'Toole
D. Rosenblitt
M. Sheldon
E. Siegal
S. Slivan
J. Stamos

Undergraduate Students

H. P. Brondmo
R. Cote
V. Diniak
R. Dreyer
D. Friedman
R. Gruber
J. Henderson
T. Huang
J. LaRocca
M. Lee
A. Sbarra
D. Segal
S. Thirugnanam
J. Woolf
K. Yeung

Support Staff

R. Bisbee

1. INTRODUCTION

The Programming Systems Research Group has been engaged in work over the past year that looks beyond present local area network assumptions, and explores new types of very large scale computer system architectures. Specifically, we have been investigating a new design for large scale distributed database systems that use database *content labels* to route queries to appropriate database servers. Within this framework we have been interested in the problems of building very large databases, both in terms of size (> 1 GByte) and in terms of geographical scope.

In order to test our architectural ideas, we have built a database system called the *Boston Community Information System* that we presently operate in the Boston area at over 200 sites. This progress report describes this system, with an emphasis on the unique way in which processing in the system is divided between a confederation of shared servers and private personal systems.

Each personal system maintains a local, user-defined subset of the database that is stored on the shared servers. Database updates are transmitted to the personal systems via a broadcast packet radio system. This design allows many queries to be completely processed at a user's personal machine, and thus reduces the reliance on shared servers.

A unifying design principle is that the system is comprised of a collection of independent personal and server databases, as opposed to a single monolithic database. Query routing is used to hide the system's division into component databases from the user. The detailed design of the shared information system and the personal database system are presented, along with reasons for our design decisions, and a review of how the design has worked out in practice.

2. PLANS FOR THE NEXT YEAR

Over the next year we plan to:

- Add new database servers, database sources, and database users to test the robustness of our present implementation.
- Gather information from the users of the database system to gain insight into the strengths and weaknesses of the system.
- Develop a new communication protocol for distributed systems that will combine the flexibility of remote procedure call with the efficiency of bulk data transport. Once we have developed this protocol, we will test it in the context of the experimental system that we have built.
- Experiment with a new approach to database services by using electronic mail as an infrastructure for database queries and responses.

3. OVERVIEW OF THE BOSTON COMMUNITY INFORMATION SYSTEM

This progress report describes a new experimental large scale community information system which was developed and is now operating in the Laboratory for Computer Science at MIT. The system is presently in use in Boston area homes, and provides users access to a number of information sources, including the *New York Times*, the *Associated Press*, and electronic mailing lists.

The system combines personal computation, broadcast digital communication, two-way communication, and centralized mass storage in a unique way. A unifying principle of the system is that it is composed of a collection of independent shared and personal databases, as opposed to one large monolithic database. Query routing is used to hide the division in function between the system's component databases.

The goals of our project look forward to the time when most information is communicated to the home and office by digital means. Today most information arrives on paper, which is intrinsically bulky. Digital delivery will make a wider range of information sources available to everyone, and computers will provide the necessary power to filter and process the large volume of information that will be received. These future systems may also be used to contribute information to public databases, as well as to conduct banking, shopping, and other transactions.

Our system design goals include:

- A cost effective way to build a system large enough to serve an entire metropolitan area;
- A high-quality user interface that is easily learned by naive users;
- Privacy for each user of the system;
- Protection, to restrict system access to authorized users;
- User autonomy, to allow users to process information drawn from the system in any way desired; and
- Flexibility, to enable new applications to be easily added to the system's existing infrastructure.

Our approach to these goals was to utilize personal computers for most processing tasks in order to minimize reliance on centralized servers. The system therefore consists of two major components: a set of shared servers operated at a central location, and a large number of personal computers, possessed by the users. The personal computers help us reach our economy, autonomy, privacy, and user interface objectives, while the shared servers provide the traditional advantages of access to a large database.

In this approach, it does not suffice to use the personal computers as mere terminals. We wanted frequent requests to be completely processed at the personal systems as often as possible. Our solution was to build a personal computer based database system with a high-performance user interface designed to operate autonomously from the shared servers. This approach seemed prudent -- although a personal computer cannot hold all of the information that might be available to a user, these computers are certainly capable of holding the information that would be of most use to a particular user. To ensure that most requests can be completely processed at the local database, a user completely specifies the information which will uniquely comprise his database's contents. When a request can not be satisfied by the personal system alone, shared servers are utilized. The use of shared servers is transparent to the user.

The integration of personal computers and shared servers into a single database environment poses a number of interesting research issues. For example, in order to ensure that our users can remain unconcerned with the mechanics of the system's operations, we chose to provide a single system image. We decided to make each database autonomous however, so that the system would be comprised of a large number of independent databases. Database content descriptions are used to route a given request to an appropriate set of databases.

Another issue that arose surrounds the requirement that large scale systems must be able to adapt to changes in infrastructures. We did not want to base our design on either specific communication channel or personal computer characteristics. As technology develops we will be able to integrate new components into our system and improve its functionality and performance.

We decided not to assume that high bandwidth communication (greater than 100 KBits/sec) is available within a community. Thus a technical challenge was to build a system that isolated users from the effects of low to moderate bandwidth communication (1.2 - 9.6 KBits/second). This was accomplished by a careful design of the user interface, to ensure that the user can proceed while communication continues in the background. The personal database system incorporates a multi-process structure which enables it to perform communication tasks while simultaneously responding to user requests.

An overview of the system is shown in Figure 8-1. The shared servers shown in the figure are located at the Laboratory. Each server contains one or more independent databases, unified by a query routing module. Some databases are replicated on more than one server in order to improve reliability and performance. As shared servers receive new information it is scheduled for transmission to the personal systems via a broadcast packet radio system. The number of times that a database item is

transmitted, and how it is protected, depend upon the item's type. The broadcast transmissions are received by the personal systems, where a background task uses them to update the local partial replica of the database. When a request can not be processed at the personal system, a connection is automatically established to the appropriate shared server.

The remainder of this report presents the components of the system, starting at the user interface and working to the shared server system. Section 4 introduces the predicate data model, which is the user's primary interface to the system. Section 5 discusses composite databases, which are used as a building block in both the personal and shared parts of the system. Section 6 presents the personal database system in detail, and provides a brief overview of its user interface. The major internal structures of the personal database system are described, along with the algorithms that are used to support query processing. Section 7 presents the internal design of the servers. The final three sections review previous work (Section 8), performance of the system and project status (Section 9), and we end with some conclusions about our approach and directions for future work (Section 10).

4. THE DATA MODEL

We have developed a new data model for storing our information which is called the Predicate Data Model. A predicate database consists of a set of records. Each record in the database is composed of a number of named fields. There are required fields (such as TYPE) and any number of optional fields. A predicate database can be mutable (records can be altered), immutable (the database can never be altered), or append only (records can be added to a database but existing records are immutable). To date we have implemented append only and immutable databases.

The fundamental operation on a predicate database is to restrict attention to a subset of the records in the database. A specification of a subset of the database is called a *query*, and the computation of a database subset is called query processing. Once a subset has been selected the records contained in the subset can be displayed or deleted.

The predicate data model provides users with a great deal of flexibility in the formulation of queries. This is important because of the character of our users. We assumed that our users know a few independent facts about the information that they seek, but are incapable of producing a unique key that would lead to the information of interest. The predicate data model permits a user to represent the facts that he or she does know about the information of interest as terms in a query.

Predicates and *result sets* are important concepts in the predicate data model. A query is a Boolean combination of *predicates*. A predicate function returns TRUE or FALSE when applied to a record, depending on the properties of the record. A record **x** is in the *result set* of a query Q iff Q(x). In our present database system, predicates include selection on the basis of:

- Record type (information source)
- Subject
- Date and time intervals
- Author
- Priority as assigned by the information source
- The appearance of user-specified words or phrases.

The predicate data model has two advantages over traditional data models, of which the second is directly relevant to the topic of this report. The first advantage is that the predicate data model is very well adapted to dealing with text and other semi-structured information that cannot be easily indexed within the framework of more traditional approaches. Research suggests that predicate based approaches may be preferred by both novice and experienced users [2].

The second advantage, and the one that is of interest here, is that the predicate data model provides a framework for reasoning about the *contents* of databases. Most traditional systems make a *closed world assumption*: if a piece of information is not found in a database, then it does not exist. This assumption of course does not apply to our system, where there are many databases, not one, and no one database contains all available information.

In order to determine where a query can be processed we must be able to reason about which database(s) contain the information which has been requested. Figure 8-4 illustrates the type of problem we are trying to solve, with two databases and three queries expressed as Venn diagrams. Query Q1 describes a potential result set that is only partially contained in DB1, but is guaranteed to be completely contained in DB2. Thus, we would process Q1 at DB2. Query Q2 describes a potential result set that is only partially contained in DB2. In this case we cannot guarantee that the result of processing Q2 at DB2 will locate all information of interest. Our approach is to have the system suggest another query to the user, Q2', which represents the intersection of DB2 and the potential result set of Q2. If the user confirms, this query can be successfully processed at DB2. The final case is represented by query Q3, which is disjoint from DB1 and DB2. This query cannot be processed, regardless of how it is extended, and is rejected.

The ability to reason in this manner about the queries presented to our system is central to our strategy for query processing. A happy consequence of the query language is that it can be directly used to formally describe the contents of databases. Let **DB** be the description of a database, phrased in terms of predicates. For example, **DB** could be a formalization of the statement that a database contains all information from the *New York Times* for March 1985.

Database content descriptions and queries are formalized in a logic that is equivalent to the statement calculus. Therefore, any statement concerning queries and database contents in our formalism is decidable.

In order to show that a query Q can be processed at a given database DB, it is sufficient to demonstrate that the statement Q(x) -> DB(x) is true. This implies that the set of records that can be described by Q is a subset of the records contained in DB. By checking the truth of statements other than Q(x) -> DB(x) we can determine when there is an intersection between Q's potential result set and DB, and when they are disjoint. When Q's potential result set is not completely contained in DB, we attempt to add the fewest number of terms possible to Q to make its potential result set a subset of DB.

The Predicate Data Model allows us to view several 'simple' databases as a single 'composite' database, as described below.

5. COMPOSITE DATABASES

Under the predicate data model, the contents of a database can be described by a boolean combination of predicates. This means that the joint contents of a *set* of databases are described by the disjunction (logical OR) of the expressions that describe the component databases. Given any query that falls within the scope of a composite database, the result set can be obtained by submitting the query to the component databases, and taking the union of the results obtained.

The expressions that describe the component databases can be used to determine *where* in the composite database certain information can be found. This gives rise to the concept of *query routing*. Rather than submitting the query to all the component databases, it suffices to submit it to any set of components for which the disjunction of the contents descriptions is implied by the query.

A similar procedure can be used to apply updates to a composite database. A given update could be applied to each component, but it suffices to apply the update to the components whose descriptions match the record.

In our system each server is implemented as a composite database. Information is assigned to component databases based on the information's type and time of acquisition. Every 24 hours the currently active databases are converted from append-only to read-only status, and a new set of databases is created. Since the databases are autonomous, they can be taken off-line without affecting the system in unexpected ways. Only the composite database manager must note the change, and update the predicate that describes the server's contents. Likewise, new databases can be added to the system without causing disruption.

From the outside a composite database is indistinguishable from a simple database. Consequently we will refer to the composite databases stored on the servers simply as 'databases'.

6. THE PERSONAL DATABASE SYSTEM

The personal database system component of our system runs on a user's personal computer, and implements the user's interface to the rest of the system. The personal database system is constantly performing two tasks: it processes user requests, and it applies database updates received over the digital broadcast channel to the local database. Figures 8-2 and 8-3 provide a view of what the user sees. Figure 8-2 shows a set of article summaries resulting from the processing of a query; Figure 8-3 shows an article which has been selected for display.

The function of the personal database system is presently limited to information retrieval. We plan to add facilities that will allow users to interactively update databases.

In order to satisfy the goals we outlined in Section 1, the system is designed so that a user's most frequent requests can be answered from the user's personal database. To this end a user compiles a list of routine queries into what is known as the filter list. Each query in the filter list is disjunctively combined to create a predicate called FL (for filter list) that describes the information that will be retained at the user's personal computer. The personal database that results is precisely the set of records necessary to completely process any query in FL, at the user's personal computer.

One limitation of this approach is that the predicate FL can describe more information than can be stored on a personal system. When this occurs the system must make a choice among the records that match FL, deciding which records to keep and which to discard. The strategy that we use for dealing with this is to have the user list the component queries of FL in order of importance, and to have the user associate a

"budget" with each query in FL. If a new record arrives that matches FL and the personal database system has insufficient resources to store it (such as disk storage or main memory) the personal database system will delete records from FL queries that are over budget. If there still is insufficient room for the new record the personal database system will delete records from the least important query in FL that is within budget.

Since the system does not necessarily always contain all the records that match FL, the system must maintain a separate predicate, PDB, that describes the local database exactly. PDB is obtained by conjoining each query of FL with an additional predicate that indicates the time elapsed since the last occasion when a record matching the query was deleted because of resource limitations.

One of the personal database system's windows displays a user's filter list. Within this window a user can easily change the filter list, or select for processing one of the filter list's component queries. Thus the user interface incorporates some of the advantages of menu based systems by allowing a user to select preplanned queries.

When a user selects a query from the filter list the system ensures that the selected query can always be processed at the user's personal system. This is accomplished by the system's taking the selected query and adding time predicates which limit the query's attention to a time interval for which the local database has a complete set of database updates. The augmented query is placed in the query input line as if the query had been typed by the user. The user is free to edit the query input line so that it includes other time periods, or he or she may type in any query at all.

Updates to the personal database continually arrive via broadcast transmission. The details of the transmission medium and packet protocol are described in another report [3].

The task of deciding how and where to process a query is called query routing. In order to perform query routing, the personal system needs the descriptions of all available databases. Based on the particular query with which it has been presented, query routing uses these descriptions to determine a feasible set of databases, any one of which could process the query. Our algorithm selects from the feasible set the database that has the lowest estimated communication cost. If the selected database is unavailable the next lowest cost database is selected.

A user's personal database system includes a description of available remote databases. To ensure that this information is always available and up-to-date, remote database descriptions are included in the database itself, and are included in the broadcast transmissions. The user's filter list predicate FL implicitly contains a term which matches records containing database descriptions. A consequence of this approach is that descriptions of remote databases are automatically kept up to date.

The overall design of the personal database system has been described, and we now proceed to present selected details about the database's implementation.

Implementation of the Personal Database System

Figure 8-5 shows the internal organization of the personal database system as implemented for the IBM PC family of computers. The modules shown are used by a collection of processes. One process reads keystrokes and mouse clicks, performs user requests, and writes to the display. A second process receives the incoming stream of database updates from the packet radio system, and applies them to the local database as necessary. A non-preemptive scheduling discipline is used. Thus, the receiver process must yield at regular intervals to give the user interface process a chance to run.

We will not describe the system's modules in detail, but we will discuss their interrelations. The window manager reads from the keyboard and mouse, passes completed commands to the database user interface, and does display maintenance as requested by the database user interface. The database user interface is responsible for implementing all commands, and for formatting database records for display. Query routing is in charge of processing queries. As described earlier, query routing uses the filter list and descriptions of remote database to decide where a query should be processed.

The local database is stored on disk, and an index is maintained in primary memory. The amount of data that can be kept in the local database depends on the amount of storage available. An average news article requires 5 KBytes of disk storage, and approximately 470 bytes of main memory. This translates to an approximate capaci[illegible] 40 news articles for a system with a single floppy disk, and 250 news articles for system with a hard disk.

Our protocol for access to remote servers is based on remote procedure call. The shared servers support the set of procedures shown in Table 1. On the personal computer side, the RPC stub shown in Figure 8-5 serves as an intermediary between these procedures, and a byte stream to a remote server. The interface was designed to include explicit flow control: each procedure invocation states how much data should be returned to the personal database system.

The RPC stub calls the reliable byte stream module in order to pass a request to a remote server. An autodial modem is used to establish a connection without user intervention. The byte stream protocol is designed to use a TCP internetwork byte stream when a local area network is available.

The reception of database updates via the packet radio system is accomplished by the set of modules at the bottom of Figure 8-5. Bytes arrive from the packet radio receiver

at 4.8 KBits/second, and are placed in a 5 KByte circular buffer at interrupt level. The size of the buffer accommodates up to a 10 second service latency. This latency can be caused by activity in the user interface process.

The receiver process polls the byte buffer and reassembles packets. The packets contain information for error detection and error correction. In order to help mask channel errors, packets are transmitted more than once, and these transmissions are separated in time. When a previously unseen packet arrives, it is copied into its proper place in the record reassembly buffer. A bit map of received packets is maintained so record reassembly can determine when an entire record has arrived.

Once an entire record has arrived, it's master key is looked up on the key ring. If a matching key is available, the record is decrypted. If the incoming record matches the filter list predicate, it is presented to the local database. The record may or may not be retained, depending on resource availability and on the record's priority relative to the information already present. The match module was designed so that the time required to match an incoming record is independent of the complexity of the filter list.

This concludes our discussion of the design of the personal database system. We now turn our attention to the design of the shared servers.

7. DATABASE SERVERS

The shared database servers in our system perform three major functions: they (1) accept data from information sources and add the data to their predicate databases, (2) transmit database updates to personal systems via a broadcast digital packet radio system, and (3) implement the remote procedure call interface described in Table 1 to allow remote access by personal systems. The organization of these functions within a single server is described below.

The conceptual organization of the data in each server borrows an important idea from the division of function between personal and server systems which was discussed in Section 3. The division between personal and server systems views data as being stored in a collection of databases, rather than in one large database. Similarly, the information in each server is organized as a collection of databases. These databases are independent in the sense that they do not share essential storage resources. For example, one database can be deleted from a server without affecting the remaining databases. The existence of multiple internal databases is not observable at the interface to the server. Query routing is used inside a server to forward a request to the proper set of databases, merge results from the databases, and formulate a response.

Implementing a server as a collection of databases allows databases to be viewed as the basic units of information that are stored on a server. This view is one level removed from individual records, and as a result provides a number of conceptual and practical advantages. These advantages include:

- The ability to view predicate databases as the basic unit of configuration, since a predicate database is described by its contents.
- The natural provision for the reconfiguration of databases from server to server, because of the predicate data model's use of database content descriptions. In our application, personal systems are automatically kept abreast, via updates transmitted over the packet radio system, of the locations of the system's databases. This capability is possible because descriptions of the contents of servers are stored in the databases themselves, and personal systems are programmed to continuously receive updates to database description records.
- The ability to achieve performance and availability goals by the replication of databases. The replication of data is most naturally done at the level of a database. Immutable databases can be easily replicated by copying them from one server to another. The replication of append-only and mutable databases is a more complicated issue.
- The ability to implement independent databases on a server in different ways while still supporting the same interface. The freedom to implement different databases in different ways is useful when the databases store different types of data. For example, most of our databases begin their lives as append only databases, and then become immutable databases after a certain amount of time. They can then be consolidated to improve efficiency and reduce storage demands.

These benefits convinced us that the decision to implement a server as a collection of databases was a good one.

Although we have discussed how data is organized in our database system, we have not examined how it got there in the first place. The bulk of our data consists of news articles, which arrive over dedicated telephone circuits. The information arrives in a constant stream, with one header per incoming news article. Each input stream is converted into a stream of database records, and the resulting records are inserted, based on their types, into appropriate databases. When a database is replicated on more than one server, the incoming telephone circuit is connected to each server so that all copies of the database are kept up to date.

A second type of input to the servers is a source that is polled for new information. In our present system electronic bulletin board entries are received in this manner. New messages are formatted into database records and inserted into appropriate databases.

As new information arrives at the system it is queued to be sent out over the digital

broadcast channel. Each record is transmitted more than once, with transmissions separated in time. Each transmitted record is encrypted to protect broadcast information from unauthorized use. The encryption key used depends on the type of the record (*New York Times, AP,* etc). This scheme logically divides the broadcast transmissions into several independently protected streams. Users are provided keys for only the streams they are authorized to receive. These keys are placed on the "key ring" of the personal database system, and enable it to decrypt transmissions.

The remote procedure call interface shown in Table 1 does not have any provision for the authentication of users that communicate with the system. We plan to have the personal database system present the keys on its key ring as passwords to enable access to independent databases.

Implementation of a Database Server

Figure 8-6 shows the block diagram of a typical server as implemented on 4.2 BSD UNIX. The exact configuration of a server depends on the databases stored on the server, the information sources it is connected to, and the communication channels that the server offers to personal systems. Only one server is used to drive the digital broadcast channel. To ensure continuity of service, a second server can take over the broadcast channel if the server running the channel fails.

In order to provide for maximum concurrent activity and failure isolation, each major function in a server is performed by a separate process. Processes communicate via messages which are placed in the file system, in an approximation of recoverable message queues. If a process fails, its input messages will collect until it is restarted. Processes do not delete their input messages until they have been completely processed. Thus, if a process fails while it is processing an input message the message will be processed again. Processes which use input messages to modify non-volatile state information must operate properly when a failure occurs. Transactions could be used to structure the failure recovery procedure [4].

In Figure 8-6, information flows from the left side of the figure to the right side. Information from the stream sources is constantly being read by input daemons that simply record incoming data items. The input daemons store independent items in separate files. The input daemons were designed to be simple, in order to reduce the chance that they would fail and lose information. To date an input daemon has never failed to properly record incoming information.

Each input daemon forwards incoming data items to a source-specific format conversion process. Each information source has a different format, and the source conversion processes take received data items and convert them into standard format database records. The resulting records are sent to the broadcast scheduler and to the database insertion process.

Figure 8-6 also shows an input process for electronic bulletin board input. This process periodically polls a set of mailboxes, and when it finds mail, converts it to standard format database records, which are then sent to the broadcast scheduler and to the database insertion process.

The broadcast scheduler process receives database records, along with an expiration time for each record, an indication of how many times the record should be transmitted, and the key by which the record should be encrypted when it is transmitted. The information broadcasts are scheduled on the basis of this information. In addition to running the digital broadcast channel, the scheduler maintains a status display for the system administrator.

The database insertion process adds, based on record type, new data items to appropriate databases. In our present system configuration the database insertion process begins to write new databases every day at midnight. The previous day's databases are no longer written and are considered immutable. Thus, there is a separate database for each type of information for each day. Each such database is kept on line for a period of time which depends on the source of the database. After this time the database is archived.

The final component of a server is the remote procedure call interface to the personal systems. When a personal system connects to a server, a process is assigned to manage the connection and perform requests from the personal system. Figure 8-6 shows one such process. Data from the personal system is first processed by a module that provides a reliable byte stream. This module is connected to the server's RPC stub module. The RPC stub module calls the appropriate procedures in the server's query routing module.

The query routing module determines a strategy, based on the descriptions of the available server databases, for processing the request. The query routing module then applies the query to the database(s) and forwards the results to the user's system as they become available. If a query spans multiple databases of a given type but with different dates, the query is applied to these databases in reverse chronological order. The most recent database records in a result set are thus produced first. The personal system receives the results as they become available. This approach often allows the system to display a complete screen of results to a user before query processing at the server is complete.

Our protocol cannot be viewed as traditional remote procedure call protocol because results are communicated to the personal system as they become available. However we have retained the idea that procedures are used to define a remote interface. This approach has worked out well in practice. The *Abort* procedure in Table 1 can be called

to force a request to abort while the request is in progress. The server will immediately acknowledge such an abort, cease work on the current request, return "ABORTED" as the result of the request that was in progress, and wait for a new request. The personal system calls *Abort* when it no longer needs the results of the request in progress.

This concludes our discussion of the implementation of the server system. The next section discusses work that is related to our approach.

8. RELATED WORK

This section compares our system to other systems which share similar goals. These other systems fall into two broad categories: systems which share our goal of being able to serve an entire community, and systems which share our goal of distributed query processing.

Community Information Systems

Teletex [7] and *Videotex* [1] are two contemporary technologies for community information systems. Teletex involves the broadcast transmission of limited amounts of information to specially equipped television sets that can display preformatted pages of information. In the teletex approach the user selects a page to be displayed, and when the page is next transmitted it is captured by the receiver in a local buffer and formatted for display on the screen. Contemporary systems limit their teletex transmissions to a total of a few hundred pages to keep response times acceptable.

Videotex is essentially time-sharing on a very large scale. Each videotex user has an inexpensive terminal which communicates with the central system over either telephone lines or digital two-way cable systems. Since videotex is a two-way system, it can also be used for home banking, shopping, and other transactional services.

In a sense, our system represents an attempt to combine the best aspects of both teletex and videotex. By combining personal computation, broadcast communication, two-way communication and centralized mass storage, our design achieves the following advantages:

- The economies of scale of broadcast communication are combined with the capability of accessing large databases and transactional services;
- A high-quality user interface is possible because of the processing power and storage available at each personal database system;
- High performance is possible because the answers to many queries are available at the user's personal computer;
- The system provides a great deal of flexibility because data can be integrated with a user's other computational tools;

- A user can access a wide variety of databases, potentially offered by a number of vendors;
- Privacy can be safeguarded by ensuring that certain requests are processed entirely within the user's personal system;
- Users can operate autonomously from any type of central system.

As mentioned above, there are two types of systems that are relevant to our discussion of previous work. The first type is community information systems, which we have just discussed. Now let us turn our attention to distributed database systems, and see how they relate to our discussion.

Distributed Database Systems

Distributed database systems, as the name implies, provide the distribution of data over a collection of cooperating computers, the resulting system can be viewed by a user as a cohesive whole. R* [8] is probably the best example of a contemporary distributed relational database system.

In R*, relations (collections of records) can be vertically distributed, horizontally distributed, or replicated at multiple sites. R* also provides a naming scheme whereby information at remote sites can be directly accessed by queries. In order to accomplish data distribution and remote access, the query compilation and query processing components of R* will automatically route queries to appropriate databases for processing. Systems similar to R* include distributed Ingress [6] and the SDD-1 system [5].

Our approach differs from traditional distributed database approaches in one important respect. In distributed database systems, dependencies can develop between databases, while in our approach databases are always considered to be strictly independent.

For many applications, distributed database systems have advantages over our multi-database approach. The model we proposed for query routing is sufficiently powerful for immutable and append-only databases. For more comprehensive systems that include replicated shared mutable data our approach could be combined with query processing strategies like those used in system R*.

However, the scheme that we have outlined is preferable in many circumstances, and has advantages including:

- Databases can be freely added to and deleted from the system without affecting other databases;
- A single simple mechanism allows our system to adjust to changes in the set of available databases;

- Queries do not have to explicitly name remote data because query routing automatically locates pertinent information.

9. PERFORMANCE AND IMPLEMENTATION STATUS

Our community information system began regular operation in April of 1984. At present 15 users outside of our own research group use a broadcast-only version of the system. The users have been very helpful in suggesting user interface enhancements (one user added mouse support himself) and motivating us to catalog the contents of our databases. In addition, the users let us know immediately if there is a problem with our database transmissions!

A research version of the system, which incorporates a preliminary form of query routing and server access, is operational, but its use is presently limited to our own group. The performance measurements shown below were derived from the research version of the personal database system. A user acceptance test of the full personal database system, including query routing and server access, is planned for this summer.

A problem with the current implementation of our system is that our software cannot collect database updates from the digital broadcast channel while a user is running other applications. We believe that this problem will be solved by the next generation of personal computer operating systems, which will support true multi-tasking.

The performance of the system is excellent for requests that can be locally processed. On an IBM AT, less than 300 milliseconds is required to either process a query or display a record from the local database.

To compare the performance of local and remote queries we performed a set of experiments. Each experiment consisted of making a request that caused the personal database system to access a server. The observed response times were directly related to the 1.2 KBit/second channel that was used for communication between the personal database system and the server. The results of our experiments are shown below:

Query entry to display of first result	2.7 sec.
Query entry to display of complete summary page	11.6 sec.
Request to display first page of article until first line arrives	4.1 sec.
Request to display first page of article until first page complete	11.9 sec.

Our system includes quite substantial databases. Every day we receive approximately 150 articles from the *New York Times* totalling 600 KBytes, 2500 articles from the *Associated Press* totalling 6 MBytes, and 70 KBytes of electronic mailing lists.

10. CONCLUSIONS

We have been pleased with the initial user acceptance of our system. Users have been able to grasp the idea of a local database that is defined by logical predicates, and we observe that users modify the definition of their local database to match their interests. Our experience has shown that broadcast technology is an effective way to update local databases. The natural advantage of the broadcast portion of our system is that it can scale to any size user population.

Database content descriptions and query routing make the personal databases provided by our system part of a comprehensive information service. Experience with our system has shown that users often want to access information which is not available on their personal system. Our system will automatically route such queries for non-local information to an appropriate server database.

In retrospect, we believe that our decision to structure our system as a collection of independent databases simplified our design, and resulted in both a clear conceptual framework and a number of practical benefits. Personal database systems will be an even more powerful component of large-scale information systems as local storage and processing continue to decline in cost. Optical storage may even allow a user to buy large databases over the counter for use on his or her personal system. Thus, the ideas that we have described for local processing and transparent remote access will be of increasing importance in future systems.

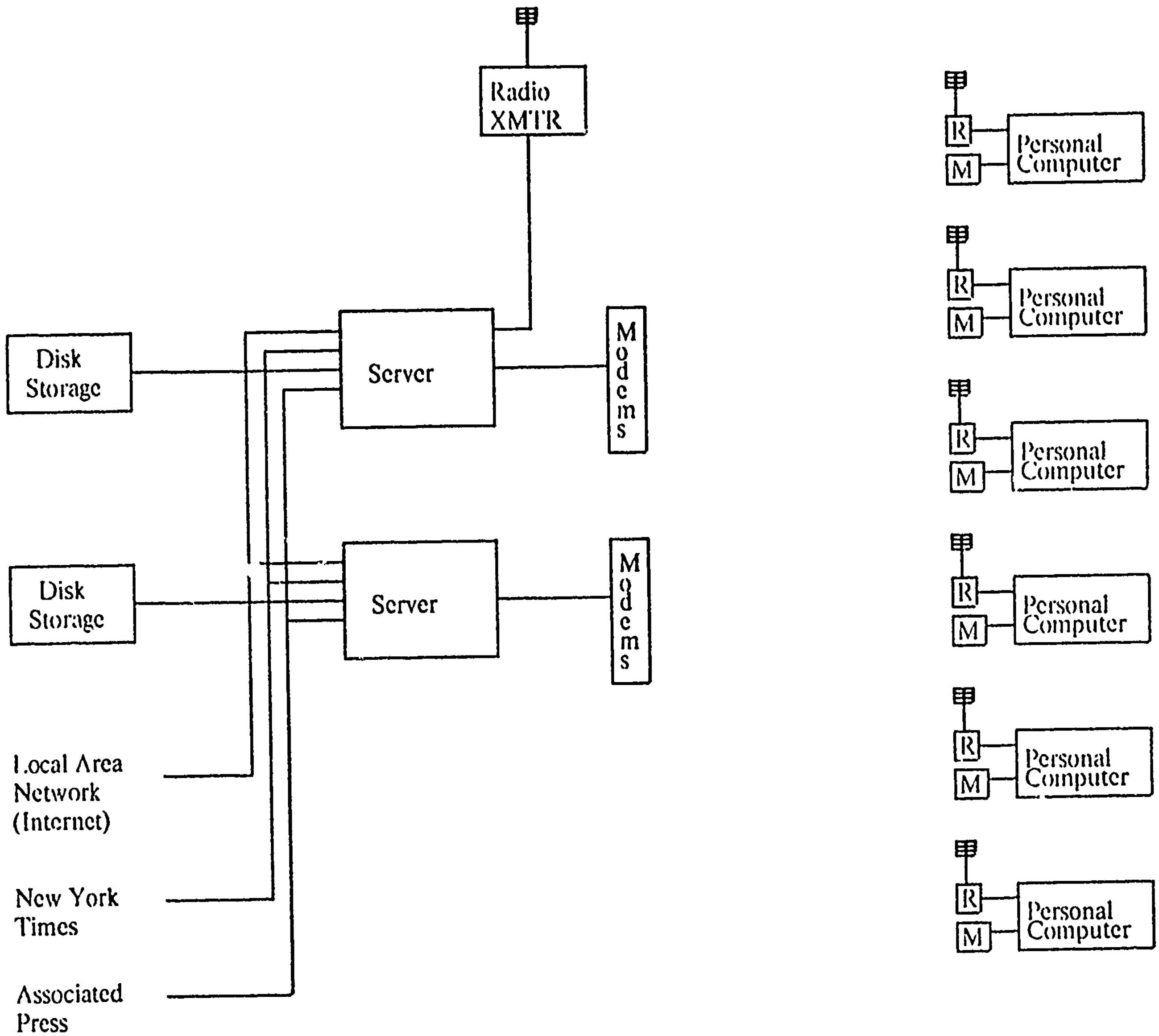

Figure 8-1: System Overview

```
 5 matching articles found.                    lines 1:18 of 18
```

```
b[ 1  sep 19, 10:48  (121 lines) regular (Financial)]
   NEW YORK -- After a record year, the market for public stock
   offerings by private companies has gone into a slump, forcing
   many of these companies to bypass the new-issue market and
   seek capital --  often through creative deals -- elsewhere.
 2 sep 18, 22:37 (80 lines) regular (Financial)
   NEW YORK -- Technology stocks took a beating Tuesday, for
   two unrelated reasons, and helped to keep the market on the
   downside.
 3 sep 18, 21:18 (82 lines) urgent (Financial)
   A digest of business and financial news for Wednesday, Sept.
   19. 1984:
 4 sep 18, 18:22 (70 lines) urgent (Financial)
   NEW YORK -- Stock prices dropped Tuesday in accelerated
   trading, with some of the technology and large
   capitalization issues registering the biggest declines.
 5 sep 18, 7:41 (113 lines) deferred (Financial)
   London - The American lawyer would have been rubbing his
   hands, except that he was jogging in Hyde Park, so he was
   swinging his arms.
```

```
 technology & (category financial);
```

Figure 8-2: Summaries

```
Article N409187.727:                                  lines 1:23 of 80
type: New York Times general news copy
priority: regular
date: 09-18-84 2237edt
category: Financial
subject: MARKETPLACE
title: (BizDay)
author: DANIEL F. CUFF
source: (c)1984 N.Y. Times News Service
text:
  NEW YORK - Technology stocks took a beating Tuesday, for
two unrelated reasons, and helped to keep the market on the
downside.
   First, worry over problems with a disk drive hurt Control
Data and Burroughs. Second, the semiconductor issues were
battered by a bearish brokerage house report on Motorola.
    Burroughs opened down 2 3/8 Tuesday morning after an order
imbalance. The drop in Burroughs, which closed the day at 53,
off 3 5/8, followed Control Data's slide. On Monday, Control
Data dropped 2 1/8, and it lost an additional 3/8 Tuesday, to
close at 26 1/8.
  Control Data, according to analysts, encountered problems with
a thin coating on the disk. ''If that chemical compound is not
virtually perfect, trouble ensues,'' said Ulric Weil, an analyst
at Morgan Stanley & Co. ''We are talking about tolerances the
thickness of a human hair.''
technology & (category financial);2
```

Figure 8-3: Selected Article

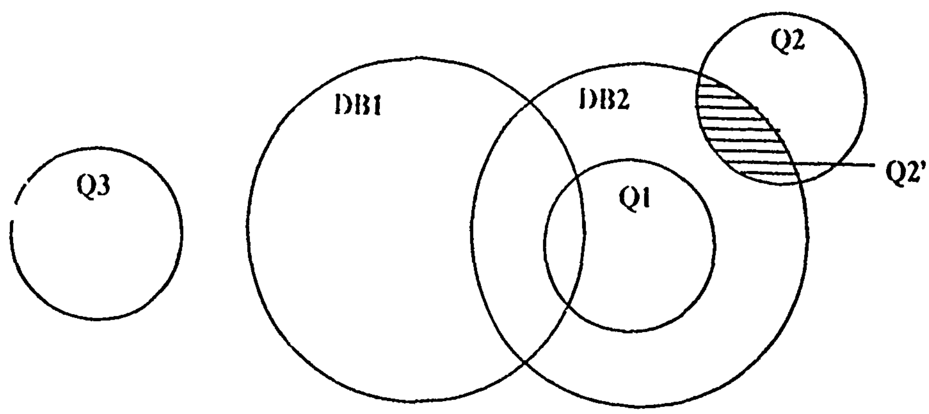

Figure 8-4: Database Descriptions and Query Result Sets

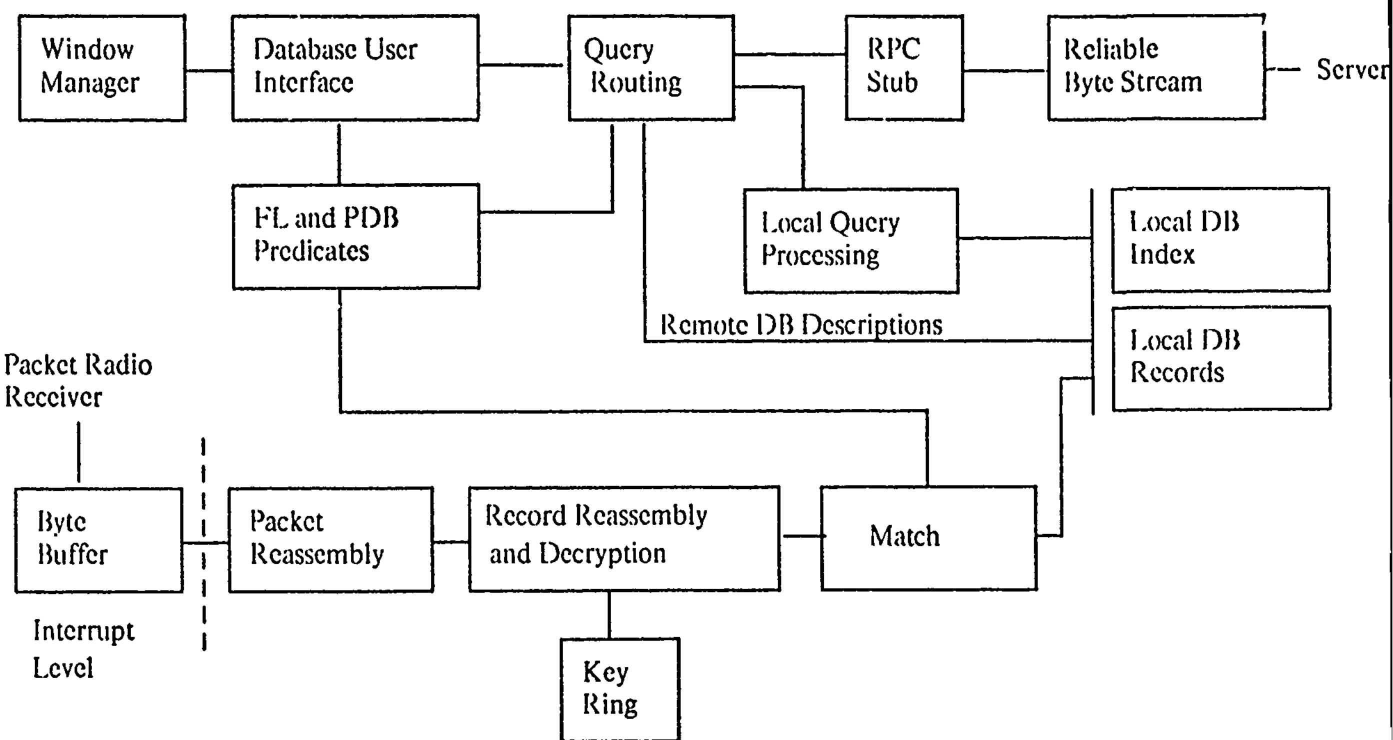

Figure 8-5: Internal Organization of the Personal Database System

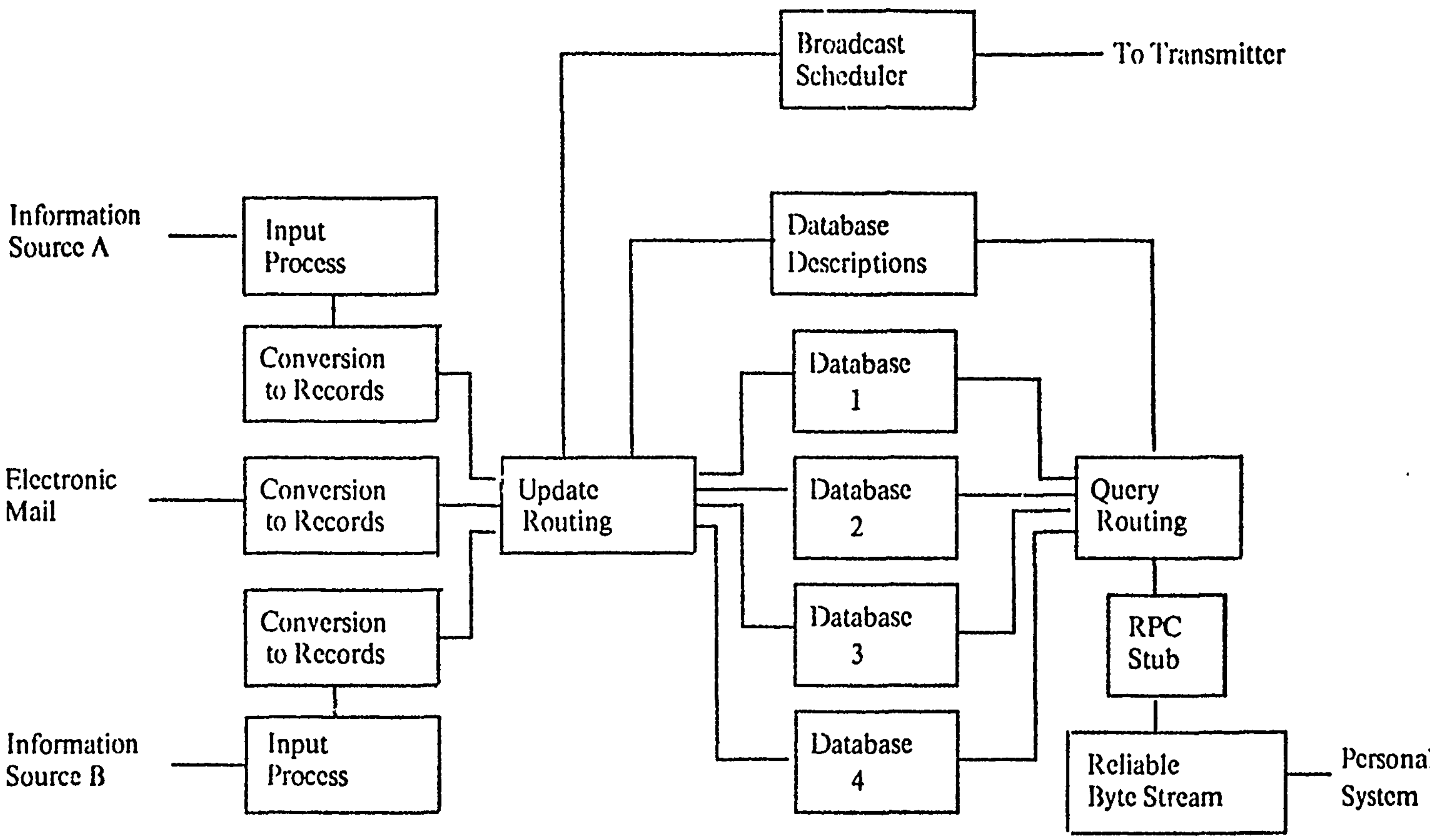

Figure 8-6: Internal Organization of a Shared Server

```
Status: TYPE = {completed, aborted, error};

Abort: PROC[]
  RETURNS [s: Status];

EstablishQuery: PROC[query: String]
  RETURNS [s: Status];

Summaries[start, finish: INT, results: PROC[s: Summary]]
  RETURNS [s: Status];

FetchRecord[record, from-line, toline: INT, results: PROC[r: Rcrd
  RETURNS [s: Status];

Matches:PROC[]
  RETURNS [final: BOOL, count: INT, s: Status];

Connect: PROC[name: String]
  RETURNS [s: Status];

Disconnect: PROC[]
  RETURNS [s: Status];
```

Table 1. RPC Interface

References

1. Bright, R.D." Prestel - The World's First Public Viewdata Service," *IEEE Transactions on Consumer Electronics* CE-25, 3 (July 1979), 251-255.
2. Geller, V.J. and Lesk, M.E. "How Users Search: A Comparison of Menu and Attribute Retrieval Systems on a Library Catalog," Bell Laboratories, Murray Hill, NJ, September 27, 1981.
3. Gifford, D.K., Lucassen, J.M. and Berlin, S.T. "The Application of Digital Broadcast Communication to Large Scale Information Systems," *IEEE Transactions on Selected Areas in Comm.*, (May 1985).
4. Gifford, D.K. and Donahue, J.D. "Coordinating Independent Atomic Actions," *Proceedings of IEEE Spring COMPCON* 85, February 25-28, 1985, San Francisco, CA, 92-95.
5. Rothnie, J.B, et. al, "Introduction to a System for Distributed Databases (SDD-1)," *ACM Transactions on Database Systems*, 5, 1, (March 1980).
6. Stonebraker, M. and Neuhold, E. "A Distributed Data Base Version of INGRESS," *Proceedings 2nd Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley, CA, May 1977, 19-36.
7. Tanton, N.E. "Teletex - Evaluation and Potential," *IEEE Transactions on Consumer Electronics CE-25*, 3, (July 1979), 246-250.
8. Williams, R., et. al, "R*: An Overview of the Architecture," Rep. RJ 3325, IBM Research, San Jose, CA, December 2, 1981.

Publications

1. Gifford, D.K. and Spector, A.Z. "A Computer Science Perspective on Bridge Design," *Communications of the ACM*, 29, 4, (April 1986), 268-283.
2. Gifford, D.K. and Spector, A.Z. "A Case Study: The CIRRUS Banking Network," *Communications of the ACM*, 28, 8, (August 1985), 798-807.
3. Gifford, D.K., Baldwin, R., Berlin, S. and Lucassen, J. "An Architecture for Large Scale Information Systems," Tenth ACM Symposium on Operating Systems Principles, Orcas Island, WA, December 1985.
4. Schroeder, M., Gifford, D.K. and Needham, R. "A Caching File System for a Programmer's Workstation," Tenth ACM Symposium on Operating Systems Principles, Orcas Island, WA, December 1985.
5. Gifford, D.K. and Lucassen, J.M. "Integrating Functional and Imperative Programming," MIT/LCS/TM-299, MIT Laboratory for Computer Science, Cambridge, MA, April 1986. (also published in Conference Record of the 1986 ACM Symposium on LISP and Functional Programming, 1986.)

6. Gifford, D.K., Lucassen, J.M., Berlin, S.T. and Burmaster, D.E. "Boston Community Information System - User Manual," MIT/LCS/TR-352 MIT Laboratory for Computer Science, Cambridge, MA, November 1985.

Theses Completed

1. Chiang, C. "Primitives for Real-Time Animation in 3 Dimensions," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1985.
2. Diniak, V.J. "Wire Service Translation Software for the Boston Community Information System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.
3. Dreyer, R. "Automatic Editing of a Data Stream," S.B. thesis, MIT Department of Electrical Engineering, Cambridge, MA, May 1986.
4. Friedman, D.A. "A Graphical Query Interface to a Cartographic Database," S.M. thesis, MIT Department of Electrical Engineering, Cambridge, MA, February 1986.
5. Martin, S. "Database Support for Cooperative Response," S.M. thesis, MIT Department of Electrical Engineering, Cambridge, MA, May 1986.
6. Stamos, J. "Remote Evaluation," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February 1986.
7. Woolf, J. "An Algorithm for the Reduction of a Cartographic Database," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

Theses in Progress

1. Baldwin, R. "Rule Based Analysis of Large Protection Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected June 1987.
2. Brondmo, H.P. "User Interface of a Dynamic Proton Synchrotron Control System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.
3. Glasser, N. "The Pipe Extension to Remote Procedure Calls," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1986.
4. Lucassen, J. "Fluent Programming Languages," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected June 1987.

5. Slivan, S. "An Interactive Graphics Interface for Clinical Applications Software," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1986.

Talks

1. Gifford, D. "An Architecture for Large Scale Information Systems," TTI Study Mission on Videotex - MIT-ILO Visit by Japanese Companies, MIT, Cambridge, MA, May 1986.
2. Gifford, D. "Electronic Publishing," Communications Forum, MIT, Cambridge, MA, April 1986.
3. Gifford, D. "An Architecture for Large Scale Information Systems," MIT-ILO Visit by Avco, MIT, Cambridge, MA, March 1986.
4. Lucassen, J. "An Architecture for Large Scale Information Systems," MIT-ILO Visit by IBM, MIT, Cambridge, MA, March 1986.
5. Gifford, D. "An Architecture for Large Scale Information Systems," 3rd Siemens - MIT Conference, Siemens Institute, Munich, Germany, February - March 1986.
6. Gifford, D. "An Architecture for Large Scale Information Systems," MIT-ILO Visit by ITT Defense, MIT, Cambridge, MA, December 1985.
7. Gifford, D. "An Architecture for Large Scale Information Systems," Tenth ACM Symposium on Operating Systems, Orcas Island, Washington, December 1985.
8. Gifford, D. "An Architecture for Large Scale Information Systems," MIT-ILO Visit by Dow Chemical Company, MIT, Cambridge, MA, November 1985.
9. Gifford, D. "Research in Database Technology," DARPA Visit, Cambridge, MA, May 1985.
10. Gifford, D. "Research in Database Technology - The Boston Community Information System," IBM/MIT Contract Meeting, MIT, Cambridge, MA, March 1985.
11. Gifford, D. and Donahue, J. "Coordinating Independent Atomic Actions," COMPCON, San Francisco, CA, February 1985.

REAL TIME SYSTEMS

Academic Staff

M. Dertouzos
R. Halstead
C. Terman
S. Ward, Group Leader
R. Zippel

Research Staff

J. Arnold
J. Pezaris
D. Robinow
R. Scheifler

Graduate Students

A. Ayres
E. Bradley
K. Dinsmore
D. Douglas
S. Gray
C. Hoffman
M. Katz
S.L. Ku
W. Lau
J. Loaiza
R. McDowell
W. Maimone
J. Marantz
J. Miller
D. Nussbaum
P. Nuth
R. Osborne
P. Osler
M. Powell
S. Seda
J. Sieber
P. Spacek
M. St. Pierre
S. Weiner
B. Williams
J. Wolfe
M. Wong
F. Zhou

Undergraduate Students

D. Becker
M. Blair
A. Braunstein
B. Byun
A. Courtmanche
L. Chao
D. Chen
S. Kukolich
S. Levitin
B.-H. Lim
J. Nguyen
K. O'Neill
A. Magnani
T. Shephard

M. Cherian
S. Clamen
M. Davon
D. Duff
G. Frazier
R. Jenez
R. Joseph
M. Singh
S. Solden
R. Spicer
N. Teagarden
D. Ward
A. Wang
P. Willison

Support Staff

A. Forestell
S. Thomas

Visitors

T. Goblick
R. Saenz
G. Thaker

1. INTRODUCTION

The 1985-6 academic year has been marked by major transitions within the RTS Group, evidenced by revision of research emphasis as well as personnel changes. The relocation of R. Halstead and his students in the Spring of 1986 constitutes the spinoff of the Parallel Programming Group (including MultiLisp and Concert) as an independent group. While related RTS activities will certainly continue to interact with and be influenced by the work o' Halstead's group, they wili presumably be separately reported in the future. We view tr..s development as an inevitable consequence of the maturity and stature which has been achieved by Halstead and his group. In addition, the imminent departure of C. Terman concludes RTS research dealing with tools for VLSI simulation. This work (whose extension to multiprocessor techniques is reported in subsequent sections) has produced a stream of widely exported results and tools during the past several years. Related work by R. Zippel, reported here, continues as a major effort in the design tools area. The L architecture, introduced in last year's progress report, has developed into a major research thrust.

2. PARALLEL PROCESSING

The past year saw continued progress in the construction of the Concert multiprocessor, now nearly complete and working fairly reliably. The software environment for Concert continued to mature as well, with the improvement of networking and diagnostic tools. The MultiLisp parallel programming language was used for several application studies, and the MultiLisp implementation also underwent some further development. A major new facility gained by the group was a collection of five Symbolics 3600 Lisp Machines, on which a very fast MultiLisp emulator has been constructed. Outside organizations have collaborated in several aspects of this work.

2.1. The Concert Multiprocessor Testbed

During the past year, Concert [5] has matured into a usable, fairly reliable testbed that is very useful as a research tool. The current Concert configuration has 24 processors organized into six clusters. Each cluster has a RingBus Interface Board, or RIB, that connects it to other clusters and globally shared memory. Eight RIBs are needed to build a full 32-processor Concert All eight have been built; only some additional backplane wiring, minor repairs to one RIB, and engineering updates to some memory boards stand in the way of building the full 32-processor Concert machine.

When running diagnostic programs, Concert operates with a mean time between errors of approximately two days. Unfortunately, some application programs, such as

MultiLisp [4], operate somewhat less reliably. When running on 24 processors, MultiLisp seems to exhibit a mean time to failure of about three hours. It is not yet clear whether this is due to hardware or software causes. In spite of this error rate and the fact that Concert only has 24 processors at this time instead of the full 32, Concert has already been used to do several significant pieces of experimental work in parallel processing. Successful projects include several application studies using MultiLisp, discussed below, and the development of the parallel logic circuit simulator PRSIM [1], discussed elsewhere in this report.

The Concert software environment has seen further development and improvement. Notably, a network server has been developed. It allows programs running on Concert to access files located on other Chaosnet hosts. It also permits much faster downloading of programs to Concert than was previously possible. Other software, especially the Concert "monitor" located in EPROM on each processor board, has been modified to fully support multi-cluster operation in a reliable and aesthetic way. The experiences of the past year have also underscored the importance of comprehensive and easily run diagnostic programs in keeping the Concert hardware operational. Consequently, significant development of diagnostics has also occurred.

2.2. Parallel Lisp

The design, implementation, and evaluation of parallel programming languages based on Lisp continues to be a major focus of research interest in the group. During the past year, members of the group have been involved in further development of environments for executing MultiLisp programs, design and implementation of Butterfly Common Lisp, and development and evaluation of several application programs written in MultiLisp.

The existing implementation of MultiLisp for Concert changed only in relatively minor ways. Several modifications were required to adapt MultiLisp for operation on multi-cluster Concert hardware. Several improvements were also made to MultiLisp. MultiLisp was modified to be able to use the network server to read and write remote files, leading to a major improvement in the convenience of performing MultiLisp experiments on Concert. Additionally, a mechanism for turning processors "off" and "on" was added to MultiLisp, greatly facilitating automated experiments in running a given program using different numbers of processors. Examples of statistics gathered using this tool are given in Figure 9-1, which reports on the performance of two parallel sorting algorithms coded in MultiLisp.

A more fundamental modification involved the addition to MultiLisp f a parcel-based parallel garbage collector, as described in [4]. This new garbage collector significantly reduces garbage collector overhead and significantly increases the size of MultiLisp computations that can be performed without free storage overflow.

As mentioned above, the reliability of MultiLisp running on Concert continues not to be as good as that of diagnostic programs running on Concert. It has been conjectured that this may be caused by the fact that 32-bit read or write operations occur in two 16-bit pieces and are not guaranteed to be atomic. Several changes in MultiLisp's way of compensating for this have been tested, but results have been inconclusive.

A major improvement in the group's computing environment came with the acquisition, beginning in August 1985, of five Symbolics 3600-class Lisp Machines. Two of the machines were purchased out of Laboratory for Computer Science equipment funding. The remaining three belong to Symbolics but have been located in 545 Technology Square for our use. Building on the efforts of D. Plummer at Symbolics, we have developed a MultiLisp emulator that runs on a Symbolics Lisp Machine. The emulator runs a subset including all the commonly used primitives of MultiLisp. The emulator also collects some data regarding the potential parallelism of emulated programs. Unfortunately, this data is subject to disruption by paging and garbage collection activity, and often it is not very reproducible. However, what the emulator loses in comprehensiveness and reproducibility of timing information, it more than makes up for in speed -- 20 to 200 times faster than a single Concert processor executing MultiLisp. This speed, as well as the extensive set of Lisp development tools hosted on the Lisp Machine, makes the Lisp Machine the environment of choice for initial MultiLisp program development and check-out, after which a program can be moved to Concert for more accurate characterization of its parallelism.

Several members of the group have been collaborating with personnel at Bolt, Beranek, and Newman in developing Butterfly Common Lisp (also known as MultiScheme), a parallel Lisp for the Butterfly parallel processor [2]. Butterfly Lisp is based on the same *future* construct and paradigm of programmer-specified parallelism as MultiLisp, but was developed by evolution from an MIT Scheme system and therefore differs considerably at lower levels of the implementation. Butterfly Lisp is operational at BBN, and can run most of the same parallel programs as MultiLisp. A sequential emulation of Butterfly Lisp runs on several Hewlett-Packard workstations at MIT.

Several studies of application programs in MultiLisp have been completed, including a simulator for clocked sequential logic circuits (Bradley), a speech analysis program to generate word hypotheses from noisy phoneme hypotheses (Lau), a set of routines for information retrieval from semantic nets (Baek), a hidden surface elimination program for display of graphic images (Jenez), and a parallel parser for Lisp S-expressions (Halstead). Results varied according to the particular algorithms and data sets used, but all application areas proved able to support at least several-fold speedups for problems of realistic size. These experiments also provided some examples where it

would be desirable to support *speculative parallelism* [6] in MultiLisp, as well as indicating some uses for a nondeterministic "choice" construct that would return whichever of a set of values is computed first, without waiting for the others. An additional benefit of these experiments is to add several more items to our library of example MultiLisp programs, which now numbers about a dozen. This library will help us calibrate the parameters of future architectures designed for executing MultiLisp.

Other work has focused on methods for automatically generating parallel MultiLisp programs without explicit specification of that parallelism by the programmer. A set of translation rules were developed for converting Lisp programs without side effects into MultiLisp programs using the *future* construct only in those places where it had some chance of generating more parallelism than overhead (Gray). The translation rules are based on a strictness analysis of the program being translated, and successfully avoid putting *future* in a large fraction (typically about 90%) of the places where it would be put by a naive *future* placement algorithm. In fact, these rules place *future* in all the places where an experienced programmer would put it, and in relatively few additional places -- perhaps 30% more, though this is highly program-dependent. Thus the translation captures almost all of the available parallelism without introducing too much avoidable overhead. These translation rules were subsequently extended to cover programs with certain restricted kinds of local side effects (Wang).

A rather different approach to parallel execution of Lisp programs was also investigated (Katz). In this approach, ordinary sequential Lisp programs are written with no explicit programmer-specified parallelism. A run-time system undertakes to exploit parallelism in the algorithm by optimistically running various parts of the program in parallel with each other and monitoring their interactions. If a conflict occurs (e.g., one task reads a value before it is written by another task that belongs at an earlier point in the original program's execution ordering), then one of the conflicting tasks is aborted and restarted. If no conflict occurs, then parallel execution has succeeded without changing the semantics of the original sequential program. Writes (side effects) are much more expensive than reads in this scheme since data can only be written provisionally by a task until its "turn" in the sequential task ordering occurs. Data written by a task cannot be permanently installed in memory as long as there is any possibility that the task will be aborted. When a write is finally installed in memory, it will cause aborts of any tasks that belong later in the sequential ordering but have already read the location being written. This proposal imposes significant bookkeeping overheads, since all reads and writes must be logged. The best way to reduce these overheads to moderate proportions is not yet clear, but the approach is interesting because it promises to allow execution of apparently sequential programs (which therefore have none of the software engineering difficulties often associated with explicitly parallel programs) with significant degrees of parallelism.

2.3. Collaboration with Other Organizations

The third full year of our collaboration with Harris Corporation has just been completed. Harris continues to build a multiprocessor testbed similar to Concert but using the GCM, a crossbar-based replacement for the RingBus, as its central communications mechanism. Debugging of the GCM is continuing. When the GCM-based Concert is operational, however, it will be largely compatible with the RingBus-based Concert, and is expected to run essentially the same software. Through the generous year-long loan of an employee (Thaker) to MIT, Harris has considerably accelerated the pace of software development for Concert, to the long-term benefit of both MIT and Harris. Other assistance from Harris has included the construction of hardware for monitoring levels of activity on the MultiBuses in Concert. Parts of this hardware have already been shipped to MIT and await installation in Concert. In return for this assistance, MIT personnel have periodically traveled to Harris and have generally assisted in developing an environment for parallel processing research at Harris. This environment is supporting the development of the PCF and Simultaneous Pascal systems for parallel programming, which are being investigated at Harris using Concert as their multiprocessor testbed.

A more substantial cooperative effort between the group and Harris Corporation barely got off the ground before becoming a victim of Gramm-Rudman budget cuts. A joint proposal was made by Harris and MIT to the DARPA Strategic Computing Program to build a second-generation multiprocessor called SCAMP. The proposal was approved and Harris was authorized to begin spending money on it. However, after several months the project was killed during a period of belt-tightening at DARPA. Currently, DARPA personnel are considering a replacement proposal from Harris and MIT that differs in various details but has similar overall goals.

As discussed above, members of the group have also been involved with personnel at Bolt, Beranek, and Newman in the design and development of Butterfly Common Lisp.

Copies of the Concert MultiLisp implementation have also been sent to the FAIM group at Schlumberger Palo Alto Research Center and to several universities.

3. THE L ARCHITECTURE

Last year's progress report introduced an effort by C. Terman and S. Ward, in its fetal stages, directed toward the development of a new non-von Neumann computer architecture. Briefly, L is a new computing model based on *object-oriented storage* and *multiple parallel threads of control.* Relying on these two constraints, L is designed to lead to efficient multiprocessor architectures that implement the advanced control and data structures popular in contemporary software. We hope that L will lead to a broad

new class of language-machine systems based on this new model of computation. L is not tied to a particular programming methodology or user-level language; however, it is specialized to programming paradigms which exploit (1) object orientation, and (2) multiple threads of control. We view L as an architectural link between multiprocessor implementations and many existing languages such as MultiLisp, Common Lisp, Ada, Prolog, Modula, and other languages yet to be invented. A technical aspiration of L is that it become a minimal model for high-level modern languages, in the same sense that RISC architectures are minimal for traditional programming languages.

3.1. Architectural Model

Unique characteristics of L include (1) its reliance on heap-allocated, fixed-size data structures called CHUNKS for ALL program and data storage; and (2) its explicit and fundamental support for computation states -- full continuations -- as first-class data objects. Within these constraints, however, L deliberately hedges on a variety of controversial issues. An example of such an issue is the extent to which run-time types are dictated by the lowest level architecture, an ongoing debate that we view as centering on interpretation versus compilation. The tagged-architecture extreme prefers run-time mechanism (demanding hardware support for efficient execution) to support types at the machine level, while (for example) the opposite extreme relies on compile-time handling of types to simplify and streamline the run-time environment. Our minimalist bias favors the latter position.

Along with contemporary compilation-intensive language advocates, however, we are unwilling to compromise the semantic advantages of run-time typing. Consequently, our data representation conventions, while almost completely dictated by compiler convention, provide data structures which are strictly self-identifying at run time. Although the L run-time conventions provide for arbitrary discriminated unions, we anticipate most type information to be compile-time constant, incurring space but not time overhead during program execution. To this end, L chunks have mandatory types which are themselves first-class L objects; typically the L type of a data structure incorporates a description of the representation conventions associated with that data structure, including sufficient information to compile accesses to the structure's components. While this type information renders the structure completely scrutible at run-time (providing, incidentally, an effective debugging environment) our expectation is that it will be rarely referenced by routine object code. The verification of this expectation (and a variety of similar ones) is, of course, among the goals of the project.

The abstract computational model engendered by L consists of a dynamic, evolving network of inter-referencing chunks, some of which are identified as representing

computation states. Each STATE chunk identifies a small set of *prerequisites,* referenced chunks whose access (exclusive or nonexclusive) is required for execution of a next instruction. At each step of the computation, some set of runnable STATE chunks with non-conflicting prerequisites is selected to be advanced to successor states. The STATE chunks so selected can then be advanced to their successor states in any order, or simultaneously, since the respective computation steps are non-interacting. The advancement of any state may lead to multiple successor states -- effectively a *fork* operation -- to be spliced into the network in the vicinity of the parent. The capacity of this approach for unbounded numbers of threads of control, together with the explicit coding of criteria for each to progress, reflects our aspiration to highly parallel L implementations.

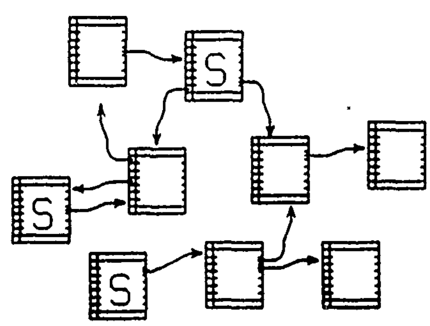

Figure 9-1: L Computation Model

Nondeterminism enters the model in that the criteria for selection of computation states to be advanced during a given step are deliberately underspecified. In a practical multiprocessor L system, that selection would be constrained by limited resources (e.g., processors) as well as by technology-dependent communication costs.

The specification of prerequisites by STATE chunks underlies the basic flow of control in the L model. A computation hangs until its specified prerequisites are satisfied; in a

typical implementation, this will happen only when the STATE chunk and its prerequisites are localized (e.g., in an execution unit). In a large system, prerequisite satisfaction may incur arbitrarily large delays which reflect the natural cost of nonlocal communications. Conversely, the L model allows such delays to be introduced artificially: a chunk ID may be introduced into a computation while the chunk it references is *locked* and consequently unavailable to satisfy prerequisites. Such an ID is, in effect, a *future* in the sense of R. Halstead's MultiLisp and provides simple access to lazy evaluation and similar programming structures.

3.2. Prototypical L Implementation

A very preliminary emulation of a single processor L engine became operational during the Spring of 1986. This implementation, written entirely in Lisp, was designed to explore fundamental architectural parameters rather than to meet even modest performance goals; it executes toy L programs at an excruciating 100 instructions per second. However, it stimulated the preliminary designs of a simulation engine [3], run-time object representations, and an assembler-level language processor. Most importantly, the first concrete instantiation of an L instruction set has established a starting point for L's evolution. Finally, this early effort set the stage for an unprecedented four to five orders of magnitude of performance improvement anticipated during the next year.

We expect to implement L entirely in microcode for the Texas Instruments Lisp Machines as our second prototype. Preliminary steps toward this goal have already been taken by the microcoded implementation of L's chunk-based storage primitives.

4. ARCHITECTURAL BUILDING BLOCKS

A number of group activities have explored architectural approaches of general interest. One such study involves a novel memory architecture in which static-column dynamic RAMs achieve the performance characteristics approaching that of a virtual cache without the expense of fast static memory. The technique is an extension of one proposed by Goodman, in which static-column RAMs are used to emulate a physical memory cache; unlike Goodman's scheme, however, our approach avoids the pagemap delay on cache hits. To demonstrate and evaluate this approach, Zak has designed and implemented a cacheless 68020 NuBus processor board capable of running a 20MHz processor out of local virtual memory without extra wait states (i.e., achieving 3-clock reads).

Various other cache-related studies have been conducted during the past year, and a number of interesting tools have been built. D. Douglas implemented a general-

purpose cache simulation tool in microcode on the Texas Instruments Lisp Machines. This tool allows a proposed cache configuration to be specified (using Lisp), then collects statistics as ordinary unmodified Lisp code is run (at a modest performance penalty). This tool was used to evaluate the static-column cache approach mentioned above in a Lisp-machine context, with quite encouraging results. Maimone developed, in TI microcode, a prototype of the metacaching mechanism proposed for use in the L architecture. His results suggest that the value of metacaching as an add-on to existing Lisp Machine architectures is limited, although we continue to speculate that it can yield dramatic results in architectures designed to exploit it.

NuBus-related engineering continues as a minor focus of the Group's activities. D. Douglas has completed a study of techniques for basing wider-word architectures on the 32-bit NuBus, using such approaches as a reblocking cache having differing word sizes at its processor and bus ports. J. Wolfe has designed and implemented a bus window useful for interfacing PC/AT compatible peripherals and processors to NuBus-based machines. Along with G. Papadopoulos of the Laboratory's MEF project, S. Ward has continued to participate in the NuBus standardization effort. Progress in that arena, although painfully slow, is apparent: the NuBus is now designated P1196 by the IEEE, and is rumored to be the basis of Apple's forthcoming "open MAC."

5. SIMULATION TOOLS FOR VLSI

The burgeoning interest in integrated circuit design has lead to an increased demand for timely, accurate circuit simulation. Due to capacity limitations of existing simulation programs, this demand is increasingly difficult to satisfy, especially as circuit sizes now routinely exceed the 100,000 transistor mark. Algorithmic improvements -- e.g., relaxation-based circuit analysis and switch-level transistor models -- have helped to close the gap, but even these simulators fall short of providing the simulation capacity required for today's large designs.

Fortunately, logic-level simulation of a large circuit actually entails the simulation of a large number of small subcircuits. With proper bookkeeping, many of the small simulations can be performed simultaneously, an observation which leads one quite naturally to the development of multiprocessor simulation algorithms. The simplicity of logic-level simulation is well matched to the computational abilities of microcomputer-based multiprocessors (which have adequate integer performance, but relatively poor floating-point abilities). On the other hand, since the simulation computations require only a few dozen instructions for each subcircuit, one must be careful that the added overhead of multiprocessor operation -- e.g., extra bookkeeping, delays due to memory contention and interprocessor communication -- does not obviate the performance increase resulting from parallel execution.

These considerations have led us (C. Terman and J. Arnold) to develop PRSIM, an algorithm based on coarse-grained parallelism, where each processor proceeds relatively independently, communicating with its neighbors only to transmit information about shared nodes. Eliminating the use of frequent, low-latency communications has the added advantage that the algorithm should perform well on a variety of different interconnect architectures; shared memory is not a requirement. As will be discussed below, reducing the amount of interprocessor communication results in some loss of efficiency since computations based on out-of-date information may need to be redone.

A few definitions will be useful below in describing how a circuit is partitioned for simulation on a multiprocessor. If one (temporarily) ignores connections made to the gates of transistors and to VDD or GND, a circuit is naturally partitioned into subcircuits called *groups*. Note that the value of a node is determined by a single group. Nodes which connect to the gates of transistors in a group are called *inputs*, and nodes in the group which control transistors in other groups are called *outputs*. For a group *g*, let *out*(g)) be the set of groups which have as an input one of *g's* outputs. In our simulation, each group is treated as a unit by PRSIM: if any input to the group changes value, new values for all nodes within the group will be computed.

For simulation on an *n* processor system, the circuit to be simulated is divided into *n* *partitions*, each composed of one or more groups. Each processor is then assigned the task of simulating one partition of the circuit. Ideally each partition could be simulated independently, keeping each of our n processors busy doing useful calculations. We may fall short of this ideal for the following reasons:

1) some partitions may take less time to simulate than others; the resulting imbalance in computation load may lead to some processors lying idle.

2) a precedence constraint between two partitions can occur when one partition determines the value of a node that is an input to the other partition. The computations of the dependent partition must always lag behind the original partition to ensure that the correct input value is used.

These problems are addressed below.

5.1. Circuit Partitioning

Processor utilization and communication costs are influenced by the manner in which the network is partitioned. To maximize processor utilization, the simulation load must be evenly distributed among the processors, which implies partitioning the circuit into pieces of roughly equal size and complexity. To minimize communications costs, one should minimize the number of links which span partitions.

For the work described here, we used a "greedy" algorithm for the initial assignment of groups to partitions, followed by a simple iterative improver which moves "edge" groups

from one partition to another. Other, more exotic, partitioning strategies are under investigation (by Davon and Frazier), but so far the improvements in simulator performance have not paid back the extra computational overhead such strategies entail. This seems especially true for large circuits where the sheer number of groups rules out visiting most groups more than once.

The greedy algorithm operates by assigning the most computationally expensive unassigned group *g* to the partition with the smallest computation load to date. Next, the members of *out(g)* are assigned to the same partition, then members of *out(out(g))*, and so on, until we exhaust this particular tree of outputs, or until the computational load for the partition exceeds the total computational load divided by the number of partitions. The whole process is repeated until all groups have been assigned to a partition.

Static partitioning algorithms, which are based on circuit topology alone, are unable to cope with the dynamic components of processor utilization and communication. Consider, for example, a simple 16-bit counter which is to be split into 4 partitions. Our algorithm would split the circuit into four 4-bit slices, balancing the apparent simulation load. The dynamic behavior, however, will be quite poor since the simulation load and communication decrease exponentially from the low order partition to the high order partition. A more effective partitioning would have placed bit 0 of the counter in the first partition, bits 1 and 2 in the second, and so on. Davon has modified our static algorithm to use profiling information accumulated during earlier simulation runs; early results on a limited number of test circuits indicate a small improvement over partitions created by the greedy algorithm.

5.2. Decoupling the Simulations

Enforcing the precedence constraint represented by a node shared between two (or more) partitions requires additional communication and can introduce delays in a poorly balanced simulation. Feedback between two partitions creates a circular precedence constraint, which can result in a forced synchronization of the simulation processes: each partition in the feedback path cannot proceed with the simulation of time step $t+1$ until all partitions have completed step t. However, if the value of each input was known for all time, there would be no precedence constraints to enforce and each partition could be simulated independently of the others. This observation suggests decoupling the partitions by introducing a *history buffer* for each partition that encodes the value of each input for all time. Simulation of a partition is then allowed to proceed based upon the assumption that the current contents of its history buffer are correct.

When a partition changes the value of an output node, it sends a message to the other

partitions for which that node is an input. The receiving partitions use this information to update their history buffers and redo their simulations if necessary. In order to correct a simulation to account for a change in the input history, we employ a checkpointing and roll back strategy derived from the state restoration approach to fault tolerance in distributed systems. As the simulation progresses, a processor saves, or *checkpoints*, all of the changes to the simulation state: inputs, outputs, and internal events. The set of checkpointed state changes forms a complete history of the simulation. The simulation of a partition can be rolled back to any previous state by walking back through the history, undoing all of the state changes.

To avoid infinite regress in feedback loops, it is sufficient to ensure that at least one partition in any feedback loop will make forward progress after each rollback. PRSIM accomplishes this by guaranteeing the propagation delay through every partition is positive, i.e., no input change at time t can produce an output change earlier than $t+1$. Thus, no partition can cause itself to roll back to the current simulated time by transmitting an output change to another partition. This ensures that each partition in a feedback loop will make forward progress on each rollback.

When the simulation of a partition must be rolled back, the state of the simulation is restored from the checkpointed information. In an earlier implementation of PRSIM, we employed a full checkpointing strategy, where the entire state of the simulation was saved periodically. To roll back, the entire simulation state was restored from the latest checkpoint prior to the input change which forced the rollback. We felt that the state of the network could be saved quite compactly (3 bits per node), and the cost of restoring the state from one checkpoint would be substantially lower than restoring from a set of incremental checkpoints. To keep both the time and storage costs of checkpointing small, the frequency of checkpoints was decreased as a function of time since the last rollback or resynchronization.

The redundant work which resulted from rolling back farther than necessary proved to be prohibitive when the number of processors exceeded about six. To solve this problem, PRSIM was rewritten to employ an incremental checkpointing strategy, in which every change to the simulation state was saved as it occurred. To roll back, each state change is undone in reverse order back to the time of the input change which forced the rollback. The state information which must be saved consists of the set of changes to node values, both internally generated events and externally generated input changes. The event list concept is extended to record past as well as future node changes by saving events after they are processed. The set of input changes to a partition is recorded in the input history buffer, described earlier. Each partition also contains an output history buffer, which records changes to nodes which are inputs to other partitions. Messages are then transmitted to successor partitions whenever changes are made to the output history buffer.

5.3. PRSIM Performance

PRSIM is a parallel circuit simulator which employs the history and roll back mechanisms presented above. As the name implies, PRSIM is based upon the RSIM algorithm. The initial implementation was designed for the Concert multiprocessor, developed at MIT. Although the Concert system is a tightly coupled parallel processor, the PRSIM algorithm does not require a shared memory architecture.

The performance of PRSIM is most easily expressed in terms of the *effective speedup*; for N processors this is defined to be $t(N)/t(1)$ where $t(N)$ is the time taken to run a given experiment on N processors. The following table shows measurements taken while simulating a 64-bit adder circuit containing 2688 transistors and 1540 nodes:

N	1	2	3	4	6
Speedup	1.00	1.86	2.43	3.11	3.77

There are three factors which prevent the effective speedup from increasing linearly with the number of processors: the extra simulation incurred as a result of roll back, the processor utilization or load balance, and the overhead associated with parallel processing. The extra simulation incurred can be expressed in terms of the *simulation efficiency*, which is defined to be n_S = events(1)/ events(N) where events(N) is the number of events processed in an N partition experiment. With up to six processors, we observed simulation efficiencies of 93%, indicating little loss due to redundant simulation.

A measure of how successfully the simulation load is balanced among the processors is the *processor utilization*, which is defined to be the percent of time each processor is busy. In the six processor experiment, the average utilization was only 71%. The low utilization accounts for most of the loss of performance in the example above; this effect may be due in part to the small size of the test circuit.

The overhead associated with parallel execution can be broken into the costs associated with the history mechanism and the costs of interprocessor communication. In the six processor experiment, 8.5% of the active time was spent sending and receiving messages, while only 3% was spent in the checkpoint and roll back mechanisms.

More extensive experiments have been performed utilizing up to 22 processors and a variety of test circuits. The results will be presented in a forthcoming paper.

6. SCHEMA DEVELOPMENTS

Schema is an integrated design system for electronic designs being developed with the help of G. Clark and M. MacDonald at Harris Corporation. There are two key problems in designing VLSI systems that Schema addresses. First, the designs themselves are quite complex and the designers need help managing the complexity of the design. Second, the design tools themselves are becoming more and more complex.

Schema addresses the design complexity issue in three ways. First, Schema provides a complete design structure in which the design can store schematics, layouts, simulation results and all other artifacts of the design. Thus it captures the *complete* design. In order to understand the impact of different design decisions, the designer needs to examine the design from different perspectives. Schema allows the designer to walk around the design laterally (through the schematics, layouts and simulation results) or vertically (from block diagram through logic and circuit schematics). Furthermore, Schema maintains consistency between the different viewpoints of the design, if possible, and marks the different components incompatible if not.

The second approach Schema uses to simplify large designs is to encourage the development of a library of *simple tools*. Simple tools perform a simple task to help the designer. Some examples are a *delay estimator* that calculates the delay between two nodes in a circuit or a power estimator that computes the average power dissipated by a set of devices. This information can be easily derived from a simulator or calculated by hand, but in either case requires enough effort on the designer's part that it will not be done often. The purpose of these simple tools is to make this information available to the designer whenever it is needed so that design decisions can be made in a confident fashion. A logic synthesizer might be another example of a simple tool. This tool would merely convert a boolean equation into a pull-down or pull-up structure. Again an easy operation for a designer but somewhat time consuming. None of these tools is a major CAD advance in and of themselves but it is our belief that cumulatively they will be.

Finally, the big tools, simulators, routers and compactors almost always have several variants. Simulations can be done using a switch level simulator, a logic simulator or a transient simulator. Routing can be done using any of a number of river routers or switch box routers. Each of these techniques is appropriate for a different segment of a design. They should be designed to be interchangeable, so that the designer can choose which one to use at different points in the design. The best example of this type

of CAD tool currently in use is a mixed mode simulator, though the decision process is controlled by the program not the designer.

To support these types of tools, the components of Schema must be more thoroughly integrated than is true of most other CAD environments. This is accomplished by the entire designing, and most design tools coexist in a common address/namespace. To encourage interchangeability in the software tools and to minimize the effort required to build CAD tools, a "Layered Language" software organization is used. Finally, a fairly complete and uniform toolbox of user interface tools is provided. Thus it is relatively rare that a tool designer need be concerned with developing a user interface to his or her tools.

In the past year we have made significant progress in Schema's development. Its schematic entry system has been vastly improved, a viable layout system is in place and the simulation tools are much improved. Each of these is described below. In addition, L. Glasser is beginning to use Schema to develop a CAD tool, the first example of a non-Schema developer building a tool with Schema. Finally, our "Layered Language" software philosophy has been more clearly articulated in internal memos and talks. For reference, the current version of Schema consists of about 1.75 million bytes of source code.

6.1. Schematic Entry System

One of the key tools in Schema is the schematic entry system. Schematics are the most common mechanism used for describing circuits, indicating how simulation should be performed and responding to user's queries. Much of the designer's time is spent dealing with them. Schema attempts to make the construction of schematics as painless as possible, enough so that the designer would prefer to sketch a schematic with Schema rather than on the back of paper napkins. This year the system has improved considerably both in performance and functionality. The functional additions allow the system to deal with busses and keep track of, and label, individual and bundled signals. An icon editor has been added which allows the designer to design icons for modules which may then be used in a schematic.

At Harris, there has been interest in using Schema to develop wire wrap and printed circuit boards. This has motivated us to develop a database of (TTL) packages that is tied to the logical functions specified via the schematic. In addition to the usual information (pin count, logical function), we chose to include speed, power, set up and hold times, and all other information that is contained in the parts data sheets. To implement our database we have chosen to use the Kandor knowledge representation language (developed by Schlumberger Palo Alto Research Center). This provides a great deal of flexibility to the designer in making queries of the package database.

6.2. Software Organization

The most common approach used in building a software system is to use a *structured design* methodology. Using this methodology, a specification for the problem to be solved is given. Using this specification, the solution of the problem is decomposed into sub-problems. This leads to a tree-like decomposition of the problem into sub-problems. To encourage modularity, each of the sub-problems is solved independently. This approach is then applied recursively to each of the sub-problems. This approach leads to a modular resolution of the problem and provides accountability for faults, an important issue in developing large software systems.

We feel that this approach is weak for a number of reasons. First and most important, invariably the problem posed is almost certainly not the problem to be solved. Either the problem's specification is incomplete or inaccurate, or by the time the system is built the needs will have changed and a slightly different system is required. Second, the modularity imposed by structured design leads to duplication of effort as each team builds the tools necessary to solve its problem. If they choose to share modules with groups working on other subproblems, then a hidden dependency will appear among the modules sharing code, reducing their modularity. Finally, the accountability benefits of structured design are really illusory. Hard problems in the resulting system are more often due to poor decomposition of the problem; thus the responsibility for the problems is collective.

In Schema, we have chosen to use what we call a "layered language" approach to building systems. To solve a problem using this approach, one generalizes the problem to be solved into a class of related problems and then develops a "language" for solving problems in this class. This new language has two components: (1) new data types and operations on the data types, and (2) new control structures (abstractions). Collectively these tools should make the solution of problems from the original class easy. The resulting system consists of many language layers, each allowing narrower, semantically richer mechanisms for describing the solution to a problem.

For instance, instead of building a single schematic entry system in Schema, we have chosen to build a language that makes it easy to construct graphics editing systems. This language includes spatial management data structures, icons and connectivity structures. New control structures exist for tracking the mouse, moving objects and synchronizing screen updates. This language is then used not only to build the schematic entry system, but also is the basis for the layout and waveform entry systems.

The new data structures are often built using the Lisp Machine's "flavor system" and the new operations using "flavor methods." The multiple inheritance and sophisticated

method combination mechanisms allow for greater sharing than many other approaches. New control structures are built using macros and functional arguments. A more detailed paper on this approach is in preparation.

7. DATABASE ACCELERATOR

Our work with Profs. Reif and Sodini on the database accelerator has been progressing very well. J. Wade and P. Osler have finished a detailed "data sheet" for the full 1024 line database accelerator. In working out the details of this specification, numerous changes were made from the original architecture proposed earlier. These changes were forced by pitch matching requirements of the final layout and pin count and power distribution considerations. A few changes were also included to improve the performance of the database accelerator in certain situations, especially when used in caching applications.

A 64-line version of the CAM has been designed and is about to be sent for fabrication. It will provide a validation of the circuit techniques used and give us some more realistic parameters before beginning the full design this fall. Furthermore, it will be used in a small (256-1024 line) board being designed by P. Osler to validate the overall approach. This small board will allow us to do preliminary studies of the microcode required by the CAM, and greatly speed up our simulations. He has completed the overall design of this board, and has implemented a custom chip that will replace the general glue logic of the board. The complete design will be done this summer.

We have also investigated the use of the database accelerator to speed up simulations. Since our CAM words are capable of holding don't cares in addition to zeroes and ones, we can use each word to represent a minterm of a logic equation. When a match operation is performed against a word with current values of the inputs, the result of the match will be the value of the minterm. Since we can match against all of the lines in the CAM in a single cycle, all the minterms are evaluated in parallel. To explore this approach B.-H. Lim has developed a simulator that makes use of the database accelerator to evaluate logic equations. Using the design goal of 90 nsec minor cycles, his results indicate the logic simulator will run at about 10 million gates per second on modest sized examples.

8. X-WINDOWS

As part of the Common System project, we are considering the adoption of the X-Window System protocol as the mechanism for imple[illegible] [illegible]ser interfaces. The desire is that any interactive application, running on an[illegible] [illegible]ne, should be able to

display its windows on any other machine. The X-Window System is fully implemented for both the C and Argus environments under Unix, but no X support for the Lisp environment on Symbolics 3600s exists.

Thus, a version of the X-Window System server has been implemented in Lisp for the 3600. This allows X programs running on Unix machines to display their windows on 3600s. An interesting feature of the implementation is its use of multiple processes; all Unix-based server implementations are single-threaded. The use of multiple processes considerably simplifies various multiplexing and demultiplexing tasks. The existing Symbolics window system is currently used to emulate the semantics of X-Windows. That is, each X-Window is implemented as a true window in the Symbolics window system. However, the two systems are fundamentally incompatible in several respects. For example, the Symbolics window system does not fully support output to partially obscured windows, but X does, and many X applications depend on this functionality. In the future we intend to reimplement the server, building a complete X environment inside a single Symbolics window. This will not only allow us to implement the X semantics precisely, but should result in significantly higher performance as well.

The other half of the problem is to allow applications written for the Symbolics window system to display on remote screens, on both Unix and Symbolics machines. Hopefully, this can be accomplished transparently to the applications, by defining a "remote screen" flavor on which application windows are instantiated. This is preferable to defining a completely new window interface, and having to rewrite existing applications to run in the new enviroriment. A design for the remote screen mechanism is currently underway, and some preliminary implementation work has been performed.

References

1. Arnold, J. M. and Terman, C.J. "A Multiprocessor Implementation of a Logic-level Timing Simulator," *Proceedings of ICCAD-85*, November 1985.

2. Crowther, W., et al. "Performance Measurements on a 128-Node Butterfly Parallel Processor," *1985 International Conference on Parallel Processing*, St. Charles, IL, August 1985, 531-540.

3. Blair, M. "A Simulator For The L Architecture in Common Lisp," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

4. Halstead, R. "MultiLisp: A Language for Concurrent Symbolic Computation," *ACM Transactions on Programming Languages and Systems*, (October 1985).

5. Halstead, R., Anderson, T. Osborne, R. and Sterling, T. "Concert: Design of a Multiprocessor Development System," *13th Annual Symposium on Computer Architectures*, Tokyo, Japan, June 1986.

6. Halstead, R. "Parallel Symbolic Computing," *IEEE Computer*, to appear, August 1986.

Publications

1. Clark, G.L. and Zippel, R.E. "Schema: An Architecture for VLSI CAD," *Proceedings of International Conference on Computer-Aided Design*, November 1985.

2. Dertouzos, M.L. "The Multiprocessor Revolution: Harnessing Computers Together," *Technology Review Magazine*, MIT, Cambridge, MA (February/March 1986).

3. Dertouzos, M.L. "Personal Computers," *1986 Edition of World Book Encyclopaedia*, Chicago, IL, 1986.

4. Halstead, R. and Loaiza, J. "Exception Handling in MultiLisp," *1985 International Conference on Parallel Processing*, St. Charles, IL, August 1985.

5. Halstead, R. "MultiLisp: A Language for Concurrent Symbolic Computation," *ACM Transactions on Programming Languages and Systems*, (October 1985).

6. Halstead, R., Anderson, T., Osborne, R. and Sterling, T. "Concert: Design of a Multiprocessor Development System," *13th Annual Symposium on Computer Architectures*, Tokyo, Japan June 1986.

Talks

1. Dertouzos, M.L. "Multiple Cooperating Computers: The New Drive in Information Technology," at Emerging Technologies: Visions and Opportunities, A Symposium for Senior Executives, Commemorating the Tenth Anniversary of the Japan Office of the MIT Industrial Liaison Program, Tokyo, Japan, March 17-18, 1986.

2. Dertouzos, M.L. "U.S. Computer Science Research: Current Thrusts and Future Opportunities," NSF Advisory Committee for Computer Research, Workshop on Planning for the Growth of Academic Computer Science, Washington, DC, February 28, 1986.

3. Dertouzos, M.L. "MIT's Project Athena: Computers in Education," MIT Club of Greece, Athens, Greece, September 24, 1985.

4. Dertouzos, M.L. Moderator: "Star Wars: Can the Computing Requirements be Met?" MIT Laboratory for Computer Science and Computer Professionals for Social Responsibility, MIT, Cambridge, MA, October 21, 1985.

5. Dertouzos, M.L. Chairman: Third Annual MIT/Siemens Joint Conference, Munich, Germany, March 1986.

6. Halstead, R.H. "MultiLisp: A Language for Parallel Symbolic Computing," INRIA, Louveciennes, France, July 22, 1985.

7. Halstead, R.H. "MultiLisp: A Language for Parallel Symbolic Computing," University of Wisconsin Distinguished Lecturer Series in Computer Architecture, Madison, WI, October 7, 1985.

8. Halstead, R.H. "MultiLisp: A Language for Parallel Symbolic Computing," Indiana University, Bloomington, IN, October 9, 1985.

9. Halstead, R.H. "MultiLisp: A Language for Parallel Symbolic Computing," University of Utah, Salt Lake City, UT, October 10, 1985.

10. Halstead, R.H. "MultiLisp: A Language for Parallel Symbolic Computing," Cornell University, Ithaca, NY, February 5, 1986.

11. Terman, C. and Arnold, J. "Prsim: A Multiprocessor Implementation of a Logic-Level Timing Simulator," Bolt, Beranek, and Newman, Cambridge, MA, June 1985.

12. Terman, C. and Arnold, J. "Prsim: A Multiprocessor Implementation of a Logic-Level Timing Simulator," Harris Corporation, Melbourne, FL, August 1985.

13. Terman, C. and Arnold, J. "Prsim: A Multiprocessor Implementation of a Logic-Level Timing Simulator," Bell Laboratories, Holmdel, NJ, October/November 1985.

14. Terman, C. and Arnold, J. "Prsim: A Multiprocessor Implementation of a Logic-Level Timing Simulator," ICCAD '85, Santa Clara, CA, November 1985.

15. Terman, C. and Arnold, J. "Prsim: A Multiprocessor Implementation of a Logic-Level Timing Simulator," Digital Equipment Corporation, Hudson, MA, March 1985.

16. Ward, S. "The L Architecture," The Third Annual MIT/Siemens Conference, Munich, Germany, March 3-4, 1986.

17. Ward, S. "The Roots of Chaos," IBM Review. February 1986.

18. Zippel, R. "Schema: An Architecture for VLSI CAD," IBM Research Center, Yorktown Heights, NY, October 18, 1985.

19. Zippel, R. "Schema: An Architecture for VLSI CAD," ICCAD '85, Santa Clara, CA, November 20, 1986.

20. Zippel, R. "Schema: An Architecture for VLSI CAD," Intel Corporation, Santa Clara, CA, November 21, 1986.

21. Zippel, R. "Schema: An Architecture for VLSI CAD," Schlumberger Palo Alto Research Center, Palo Alto, CA, November 22, 1986.

22. Zippel, R. "Schema: An Architecture for VLSI CAD," Industrial Liaison Program Symposium, Santa Clara, CA, January 29, 1986.

23. Zippel, R. "Schema: An Architecture for VLSI CAD," Texas A & M University, College Station, TX, April 16, 1986.

24. Zippel, R. "A New Approach to CAD Software in Schema," Texas Instruments, Dallas, TX, April 17, 1986.

25. Zippel, R. "A New Approach to CAD Software in Schema," University of California, Berkeley, CA, May 7, 1986.

26. Zippel, R. "A New Approach to CAD Software in Schema," Stanford University, Palo Alto, CA, May 8, 1986.

27. Zippel, R. "The MIT Database Accelerator," Stanford University, Palo Alto, CA, May 9, 1986.

Theses Completed

1. Baek, H. "Parallel Retrieval Algorithms for Semantic Nets," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, April 1986.

2. Blair, M. "A Simulator For The L Architecture in Common Lisp," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

3. Bradley, E. "Logic Simulation on a Multiprocessor," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

4. Byun, B. "A Cached File System for Distributed Workstations," S.B. thesis,

MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

5. Chen, D. "DYALOG: A Functional Block to Gate Level Compiler for Semi-Custom Integration," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986

6. Clamen, S. "Debugging in a Parallel Lisp Environment," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

7. Courtmanche, A. "MultiTrash, A Pa. 'ıl Garbage Collector for MultiScheme," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February 1986.

8. Davon, M. "Partitioning Digital LSI Circuits for Parallel Simulation," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

9. Dinsmore, K. "Performance Measurement Techniques for Multiprocessor Systems," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

10. Douglas, D. "A High Performance Interface for a 40-Bit Machine on the NuBus," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

11. Duff, D. "A Computer-Aided ALU Topology Generator," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

12. Frazier, G. "Analysis of Algorithms to Partition NMOS Circuit Designs for Testing Programs," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

13. Granger, L. "Lisp-Based Tests and Diagnostics for the M.E.F. Circuit Switch," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

14. Gray, S. "Using Futures to Exploit Parallelism in Lisp," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

15. Joseph, R. "An ECL Implementation of a RISC Architecture," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

16. Katz, M. "Paratran: A Transparent Transaction Based Run Time Mechanism for Parallel Execution of Scheme," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

17. Kukolich, S. "Towards a Machine Independent Compiler System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

18. Lau, W. "Lexical Analysis of Noisy Phonetic Transcriptions," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February 1986.

19. Levitin, S. "MACE: A Multiprocessing Approach to Circuit Extraction," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

20. Lim, B. "Logic Simulation Using a Content-Addressable Memory System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

21. Magnani, A. "A VLSI, Sub-Ten Nanosecond, 32-Bit Carry Chain for a Full Adder," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

22. Maimone, W. "Metacaching in Object Oriented Architectures," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

23. McDowell, R. "Architecture-Specific Code Optimization with a Machine-Independent Code Reorganizer," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

24. Osborne, R. "Modeling the Performance of the Concert Multiprocessor," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

25. Shephard, T. "A Hardware Simulator Employing a Charge-Based Model for MOS Digital Circuits," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

26. Solden, S. "Implementing Signal Wave Forms as First-Class Objects in CAD Tool, Schema," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

27. Spicer, R. "Simulation of Digital MOS Circuits via Charge Analysis," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

28. St. Pierre, M. "A Simulation Environment for Schema," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

29. Teagarden, N. "Porting Schema to the Texas Instruments Explorer," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

30. Wang, A. "Exploiting Parallelism in Lisp Programs with Side Effects," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

31. Ward, D. "Digital Telephone Answering Unit," S.B. thesis, MIT Department

of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

32. Willison, P. "Lisp Data Structure for the "L" Architecture," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

33. Wong, M. "A Combining Omega Network: Performance vs. Implementation," S.B. and S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1986.

Theses in Progress

1. Ayres, A. "TBA" S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1987.

2. Hoffman, C. "Linguistic Foundations for an Algebra System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1986.

3. Ku, S.-L. "A High Level Microcode Development System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1986.

4. Ma, M. "Efficient Message-Based System for Concurrent Simulation," Ph.D dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1987.

5. Miller, J. "Multischeme: A Parallel Processing System Based on MIT Scheme," Ph.D dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1987.

6. Nussbaum, D. "A Framework for Real-Time Graphics on a Multiprocessor," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.

7. Nuth, P. "A Processor for MultiLisp," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.

8. Osler, P. "A Prototype Content Addressable Memory System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1986.

9. Powell, M. "To Be Determined," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1987.

10. Seda, S. "Compacting VLSI Layouts Containing Octagonal Geometry," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1986.

11. Sieber, J. "To Be Determined," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1987.

12. Spacek, M. "A Network-Based Real-Time Interactive Simulation," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.

13. Williams, B. "Principled Design Based on Qualitative Behavioral Descriptions," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1987.

14. Wolfe, J. "TI NuBus to IBM PC/AT Bus Converter," S.B. and S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1986.

SYSTEMATIC PROGRAM DEVELOPMENT

Academic Staff

J. V. Guttag, Group Leader

Research Staff

D. Detlefs
S. J. Garland

Graduate Students

D. S. Hinman
J. L. Zachary
K. A. Yelick

Support Staff

A. L. Rubin

1. INTRODUCTION

The Systematic Program Development Group is currently working on two related projects, the Larch Specification System and the Reve Term Rewriting System. The work on Larch has been supported primarily by DARPA, and the work on Reve primarily by the NSF.

2. LARCH

The Larch Project is a collaborative effort between MIT's Laboratory for Computer Science and DEC's Systems Research Center. The project is developing both a family of specification languages and a set of software tools to support their use. The support tools include language-sensitive editors and semantic checkers based on a powerful theorem prover. The goal of the project is to explore the application of precise specifications to the development and maintenance of software. Several assumptions have influenced the directions taken by the Larch Project.

- *Local specifications.* Behavioral specifications of program modules will be useful in the near future. No conceptual breakthroughs or theoretical advances seem to be needed. Rather, we need to use what we already know to design usable languages, develop software support tools, and educate system designers and implementors.

- *Error detection.* The process of writing specifications can be as error-prone as the process of programming. Therefore, it is important to check specifications thoroughly for errors. There are two ways to detect errors: human inspection and mechanical analysis. The Larch languages are designed to make it easy to write readable specifications, and formal specifications written in Larch have an advantage over informal specifications in that they can be checked mechanically for many common errors.

- *Scale.* Specification methods must remain useful even when there are many requirements to be recorded in a specification. Methods that are entirely adequate for one-page specifications may fail utterly for hundred-page specifications. It is essential that large specifications be composed from small ones that can be understood separately, and that the task of understanding the ramifications of their combination be manageable.

- *Incremental construction.* Large specifications, like large programs, must be constructed incrementally. Furthermore, many of the benefits of any kind of specification come during the process of constructing the specification: the very act of specification sheds light on what is being specified. The process of careful specification encourages prompt attention to inconsistencies, omissions, and ambiguities; and it often results in a better understanding of the problem.

- *Incompleteness.* Most of the important analyses of specifications must

occur while they are still incomplete. Most specifications are unfinished during most of their useful lifetime. Consequently, it is essential to reason about and to check unfinished specifications. Many finished specifications are still incomplete. Sometimes this incompleteness is caused by abstraction from details that are irrelevant for a particular purpose; for example, time, storage usage, and functionality might be specified separately. Sometimes, however, it is a symptom of an oversight in the design or specification process. A specification checker should be able to distinguish between oversights and intentional incompleteness.

- *Tools.* Tools have an important role to play in the specification process. The number of tedious and error-prone tasks associated with maintaining a substantial body of formal text in a consistent state is a serious bar to the practical use of formal specifications. Tools can assist in managing the sheer bulk of large specifications, in browsing through selected pieces, in detecting errors, in deriving interactions and consequences, and in teaching a new methodology. Languages designed to exploit powerful tools may be quite different from pencil-and-paper languages.

- *Reusability.* It is inefficient to start each specification from scratch. We do not want to keep reinventing the specifications of integers and sets - or even of priority queues and bitmaps. We need a repository of reusable specification components that have evolved to handle the common cases well, and that can serve as models when we are faced with uncommon cases. The collection should be open-ended and include application-oriented abstractions, as well as mathematical and implementation-oriented ones.

- *Language dependencies.* The environment in which a program is embedded, and hence the nature of its observable behavior, is likely to depend in fundamental ways on the semantic primitives of the programming language. Any attempt to disguise this dependence will make specifications more obscure to both the component's clients and its implementors. On the other hand, many of the important abstractions in most specifications can be defined independently of any programming language.

- *Implementation independence.* A good specification strikes a careful balance between restrictiveness and generality. It tells the implementor what service to provide, but does not place unnecessary constraints on how it is provided. In this way it allows the implementor as much flexibility as is consistent with the users' requirements. And it tells users just what they can expect from a module, including how that module deals with exceptions and boundary conditions. Without a specification, all that exists is the code, and it is unclear what effects modifications to the code will have on its behavior.

2.1. The Larch Family of Specification Languages

The Larch Family of Specification Languages is based on a *two-tiered* approach to specification. Each Larch specification has components written in two languages: one designed for a specific programming language, and another common to all programming languages. We call the former *Larch interface languages*, and the latter the *Larch Shared Language*.

The Larch Shared Language is used to specify language-independent abstractions that are useful for specifying module interfaces. Its role is similar to that of *abstract models* in some other styles of specification. It has a simple semantic basis and allows substantial checking. An introduction and reference manual [3], and a large set of examples [4], have been published.

Each Larch interface language deals with what can be observed about the behavior of programs written in a specific programming language. Its simplicity or complexity is a direct consequence of the simplicity or complexity of the observable state and state transformations of the programming language. Descriptions of two interface languages, Larch/CLU and Larch/Pascal, are contained in [5]. Two further interface languages, one for concurrency and one for Modula-2, are under development and are discussed below.

It is too soon to draw any strong conclusions about the utility of Larch in software development. We have written a significant number of Larch Shared Language specifications. On the whole, we were pleased with the specifications, and with the ease of constructing them. While writing them, we uncovered several design errors by inspection; we are encouraged that many of these errors would have been uncovered by the checks called for in the language definition. However, until we have more experience with automated semantic checking, it is impossible to know how helpful these checks will be.

We have just begun to write substantial specifications in Larch interface languages. Our experience leads us to be optimistic. In particular, we have been pleased with the Larch style of two-tiered specification. Several benefits arise from our ability to factor Larch specifications, both into shared and interface components, and also into separate traits and module specifications within these components. We find that parts of a factored specification are easier to reuse, and that an entire specification is easier to comprehend since its novel features are clearly distinguished from familiar features present in related specifications.

2.2. An Editor for the Larch Shared Language

Over the last few years we have developed a systematic approach to designing and writing Larch Shared Language specifications. In particular, we have designed and implemented a syntax/semantics-directed specification editor that guides the user through the specification process. The editor provides the following capabilities:

- *Templates.* The Larch editor generates templates for the components of a specification. To write a specification, the specifier simply fills in these templates. Templates remaining in an incomplete specification remind the specifier of what information is required to complete the specification or to resolve inconsistencies in the specification.

- *Flexible notation.* Most programming and specification languages, together with their associated editors, provide a fixed set of symbols (such as + or *) that can be used as infix operators; users of these languages can introduce other operators, which they must write using prefix notation. The Larch editor does not prejudge which symbols specifiers will want to use. Instead, it allows them to introduce operators using *mixfix* notation; thus specifiers can introduce natural notations such as $<x,y>$ for an ordered pair or $A[n]$ for an element of an array.

- *Redundant information.* The Larch editor generates redundant information that makes a specification easier to read. Some of this information takes the form of "syntactic sugar" that makes a formal specification look more like ordinary English. Some takes the form of information replicated at crucial points throughout a specification. In all cases, the Larch editor relieves specification writers of the burden of typing extra information for the benefit of specification readers.

- *Error prevention and detection.* Our syntax-directed approach to editing prevents errors in the format of a specification. The Larch editor also supplies automatic type and semantic checks that help specifiers catch other mistakes early. Experience indicates that seemingly superficial errors are often indicative of serious underlying confusion, and catching them early is a valuable service.

- *Automatic inferencing.* Most programming languages and language processors use declarations supplied by the user to detect errors in a program. Yet programmers often find it tedious to create the redundant information required by these declarations. The Larch editor aids specifiers by inferring, insofar as possible, declarations for variables and operators from the use of those variables and operators.

This past year, S. Garland has increased the capabilities of the Larch editor and improved its user interface so that it is now a useful tool for creating specifications.

2.3. Analysis Tools

The design of the Larch family of specification languages was strongly influenced by the goal of checking specifications. We decided that it would be a mistake to rely only on the syntax and type checking found in many "implementations" of specification languages. Checks based on the meaning of specifications seemed crucial. Towards this end we have developed a state-of-the-art theorem prover based on term rewriting systems (see the following discussion of Reve).

At present, our system (Reve 2.4) enables us to check the consistency of an equational theory associated with a specification. Next year, we plan to implement additional aids that will help check whether two specifications are independent of each other, or whether the action of an operator has been specified fully.

Our approach to theorem proving is based on the goal of limiting, as much as possible, interaction between the theorem prover and its users. This is a consequence of our desire to embed the theorem prover in another program that analyzes specifications. Users of the specification analyzer should be spared the trouble of having to learn very much about how the theorem prover works. The goal of limiting user interaction with the theorem prover leads us to trade machine time for user time.

Where possible we want to perform our analyses in "real time" so as to alert the specifier to anomalies as soon as possible. It is particularly useful to alert the specifier immediately to an inconsistency between the part of the specification currently being written and parts of the specification supplied earlier. Failure to do so increases the likelihood that the specifier will waste considerable effort traveling down an unproductive path. To this end, we have acquired a Vax 8600 for use as a theorem-proving server.

2.4. Concurrency

This past year, J. Guttag has worked with J. Horning at DEC's System Research Center to define and use a new Larch/Concurrent interface language. Our approach has been to handle concurrency, which is implemented in different ways on different systems, in an interface language. We have been pleased that the design of the Larch Shared Language has supported this work on concurrency without any need for modification.

In specifying concurrency, we make a distinction between atomic and non-atomic actions. The key property of an atomic action is that of serializability: for every concurrent execution of a set of atomic actions there exists a sequential execution of the same actions with the same net effect. An atomic procedure, which executes a single atomic action, is similar to a sequential procedure in that states between the

invocation of a procedure and its return cannot be observed. Therefore we can specify them using pre- and post-conditions corresponding to the start and finish of the atomic action.

Many procedures in concurrent environments are inherently non-atomic, and we specify them as a fixed sequence of actions. These actions may be named; in particular, the initial and final actions are given reserved names. The specification thus consists of a set of conditions on the actions. The following example specifies one condition that holds for all actions between the first and final action, and one condition that holds only the final action:

procedure `DoUpdate( d: UpdateSet, us: UpdateSet )`
modifies at most `[d]`
each action ensures
$d_{pre} \subseteq d_{post} \subseteq d_{pre} \cup us$
final action returns ensuring
$us \subseteq d$

We have applied the Larch/Concurrent interface language to specifying the synchronization mechanisms for Topaz environments and to specifying a global name server. Writing these specifications guided the design of the interface language and also provided some insight into the practical difficulty of specifying concurrency. The first specification is complete and currently in use, and the second is still in progress.

In both cases, the Larch/Concurrent specifications proved to be more comprehensible, and to place fewer unnecessary restrictions on implementations, than did previously existing operational specifications. They also uncovered previously undetected errors in the operational specifications.

2.5. Larch Interface Languages

Larch interface languages are programming-language dependent. Everything from the modularization mechanisms to the choice of reserved words is influenced by the programming language. At present, there are two moderately well-developed Larch interface languages, Larch/CLU and Larch/Pascal. A detailed description of the semantics of an early version of Larch/CLU is given in [12]. A detailed description of Larch/Pascal has yet to be written. However, a discussion of the style of Pascal programming that Larch/Pascal is designed to support is contained in [9].

While the semantic bases of Larch/Pascal and Larch/CLU are considerably more complicated than that of the Larch Shared Language, their static semantics are considerably simpler. In the Shared Language, there are a large number of mechanisms for building one specification from another and for inserting checkable

redundancy into specifications. Corresponding mechanisms are not present in interface languages. We wish to encourage a style of specification in which most of the programming-language-independent complexity is pushed into the Larch Shared Language. We feel that specifiers are less likely to make serious mistakes in this simpler domain. Furthermore, it is easier to provide machine support that will help specifiers catch the mistakes that they do make. Finally, by encouraging specifiers to put effort into Shared Language specifications, we increase the likelihood that parts of specifications will be reusable - not only for different specifications written in the same interface language, but also across specifications written in different interface languages.

The semantic bases of Larch interface languages tend to mirror the semantic bases of the programming languages from which they are derived. In general, this means that the semantics of an interface language is rather complex. D. Hinman is working on the design of a Larch/Modula-2 interface language and on a formal definition for the semantics of this interface language. His work will provide a model for defining the semantics of other interface languages, being less tied to the specific semantics of the underlying programming language than was the earlier formal treatment of the Larch/CLU interface language.

3. THE REVE TERM REWRITING SYSTEM

The availability of automatic tools for reasoning about Larch specifications is essential to our specification approach, and has influenced the design of Larch itself. Accordingly, we have devoted considerable effort to the further development of Reve [2], a theorem prover for equational theories. We are currently using Reve to verify the consistency of specifications written in the Larch Shared Language. Reve is also in use in approximately thirty university and industrial laboratories in the United States and Western Europe.

3.1. Reve 2.4

The Reve Term Rewriting System includes a robust implementation of the Knuth-Bendix [7] completion procedure, partial support for inductionless induction using the Huet-Hullot method [6], and a laboratory environment for rewriting and unifying terms. The current release, Reve 2.4, of Reve incorporates the following new features:

- *Associative-Commutative Operators.* Reve 2.4 handles associative-commutative operators in a fully integrated fashion. Its matching, unification, and completion algorithms automatically check for these operators and invoke the proper procedures to handle them. It employs a new unification procedure that is based on the work of K. Yelick (see

below); its completion procedure is based on that of Peterson and Stickel [11]. Using Reve 2.4, we have been able to complete many sets of equations that could not be completed previously in Reve.

- *Orderings.* In Reve 2.3, the user interface to Knuth-Bendix was greatly simplified by the development of new ordering procedures. These procedures compute all the ways in which an ordering (used to prove the termination of a rewriting system) can be extended to order a new equation into a rewriting rule. Thus, where the RDOS [8] of Reve 2.2 required the user to be fairly knowledgeable about orderings, the RDOS of Reve 2.3 and Reve 2.4 presents the user with a complete list of possible actions [1]. Reve 2.4 also provides a new ordering based on polynomial interpretations of operator symbols. This ordering is significant because it provides one of the few methods that can be used to prove the termination of rewriting systems for associative-commutative operators.
- *Automatic Orderings.* Our ability to compute a complete set of extensions for an ordering allows us to decide whether that ordering can be extended to complete a set of equations, and it allows us to construct automatically the extension if it exists. This automatic ordering greatly reduces the amount of interaction needed to complete a set of equations. In the worst case, it must search a set of extensions that grows exponentially with the number of operators in the rewriting system. But Reve 2.4 uses several techniques designed to reduce the average time required to order a set of rules, and extensive experimentation has convinced us of the practical utility of our implementation.
- *Critical Pairs.* Reve 2.4 uses a new method for deciding what pair of rules to use next to compute critical pairs. It keeps a list of pairs of rules for which critical pairs have not yet been computed. When a request comes for the next pair of rules to use, this list is sorted by some metric (currently the combined size of the left-hand sides) and the smallest is returned. In some completions involving large numbers of rules, this can result in significant savings. It is also conceptually simple and easy to change by changing the sorting criterion.

We plan to extend Reve by adding unification and matching algorithms for equational theories other than the empty theory and the associative-commutative theories. Our immediate plans are driven by the need to handle equations that have arisen in attempts to use Reve to reason about specifications of programs.

3.2. Unification

K. Yelick [13] has designed a method for combining equational unification algorithms to handle terms containing "mixed" sets of function symbols. For example, given one algorithm for unifying associative-commutative operators, and another for unifying commutative operators, her algorithm provides a method for unifying terms containing both kinds of operators. She proved that the method is consistent, complete, and terminating for a class of theories called the confined regular theories.

She is working jointly with J. Zachary to extend her unification algorithm to combinations of sorted, rather than just unsorted equational theories. By making use of sort information, they can relax some of the restrictions on the equational theories and further optimize the unification process. She is also working with J.-P. Jouannaud of the Universite de Nancy on equational unification for data types.

J. Zachary is continuing an investigation into the pragmatics of incorporating data abstraction into the paradigm of logic programming. His research has focused upon the design of a new programming language, called Denali, that provides data abstractions similar to CLU clusters. Whereas a CLU cluster defines a set of abstract objects and provides a set of operations to manipulate these objects, a Denali cluster defines a set of terms and provides a set of predicates to manipulate these terms. The predicates are defined using Prolog-style Horn clauses.

Each Denali cluster provides a means of unifying pairs of the terms that it defines. In Prolog, all terms are unified using classical unification [10] because no interpretation is placed upon the function symbols that compose them. Abstract data types, in contrast, associate meanings with the function symbols they introduce. For example, the two terms x+3 and 5, while not unifiable in the classical sense, can be unified by the substitution <*x* gets 2> when regarded as integers.

Although an implementation of Denali can provide unification algorithms for built-in abstractions, the programmer is responsible for coding algorithms that unify pairs of user-defined terms. One of Zachary's goals is to make Denali powerful enough to express these user-defined unification algorithms. Since it is a difficult problem in general to devise unification algorithms, Denali provides several features that render the programmer's task more tractable. For example, Denali clusters can provide restricted sets of terms, or they can restrict the pairs of terms they will unify; such restrictions can make it easier to develop an appropriate unification algorithm.

References

1. Detlefs, D. and Forgaard, R. "A Procedure for Automatically Proving the Termination of a Set of Rewrite Rules," *Proceedings of the First International Conference on Rewriting Techniques and Applications*, Dijon, France, May 1985.

2. Forgaard, R. and Guttag, J. V. "REVE: A Term Rewriting System Generator with Failure-Resistant Knuth-Bendix," *Proceedings of an NSF Workshop on the Rewrite Rule Laboratory*, General Electric Corporate Research and Development Report No. 84GEN008, Schenectady, NY, April 1984, 5-31.

3. Guttag, J. V. and Horning, J. J. "Report on the Larch Shared Language," *Science of Computer Programming*, 6, (April 1986), 103-134.

4. Guttag, J. V. and Horning, J. J. "A Larch Shared Language Handbook," *Science of Computer Programming*, 6, (April 1986), 134-157.

5. Guttag, J. V., Horning, J. J. and Wing, J. M. "An Overview of the Larch Family of Specification Languages," *IEEE Software*, (September 1985), 24-35.

6. Huet, G. and Hullot, J. M. "Proofs by Induction in Equational Theories with Constructors," *Proceedings of the 21st Symposium on the Foundations of Computer Science*, Los Angeles, CA, October 1980, 96-107.

7. Knuth, D. E. and Bendix, P. B. "Simple Word Problems in Universal Algebras," in *Computational Problems in Abstract Algebra*, J. Leech (ed.), Pergamon Press, Oxford, 1969, 263-297.

8. Lescanne, P. "Uniform Termination of Term Rewriting Systems: Recursive Decomposition Ordering with Status," *Proceedings of the 6th Colloquium on Trees in Algebra and Programming*, Bourdeaux, France, Cambridge University Press, March 1984. Also appears as "How to Prove Termination? An Approach to the Implementation of a New Recursive Decomposition Ordering," *Proceedings of an NSF Workshop on the Rewrite Rule Laboratory*, General Electric Corporate Research and Development Report No. 84GEN008, Schenectady, NY, April 1984, 109-121.

9. Liskov, B. H. and Guttag, J. V. *Abstraction and Specification in Program Development*, Cambridge, MA, MIT Press, 1986.

10. Robinson, J. A. "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of the ACM*, 12, 1, (1965), 23-41.

11. Stickel, M. E. "A Unification Algorithm for Associative-Commutative Theories," *Journal of the ACM*, 28, 3, (July 1981), 423-434.

12. Wing, J. M. "A Two-Tiered Approach to Specifying Programs," Ph. D. dissertation, MIT/LCS/TR-299, MIT Laboratory for Computer Science, Cambridge, MA, May 1983.

13. Yelick, K. A. "Combining Unification Algorithms for Confined Regular Equational Theories," *Proceedings of the First International Conference on Rewriting Techniques and Applications*, Dijon, France, May 1985.

Publications

1. Forgaard, R. "A Program for Generating and Analyzing Term Rewriting Systems," MIT/LCS/TR-343, MIT Laboratory for Computer Science, Cambridge, MA, August 1985.

2. Garland, S. J. *Introduction to Computer Science with Applications in Pascal*, Reading, MA, Addison-Wesley Publishing Company, 1986.

3. Garland, S. J. *Instructor's Guide for Introduction to Computer Science with Applications in Pascal*, Reading, MA, Addison-Wesley Publishing Company, 1986.

4. Guttag, J. V. and Horning, J. J. "Report on the Larch Shared Language," *Science of Computer Programming*, 6, (April 1986), 103-134.

5. Guttag, J. V. and Horning, J. J. "A Larch Shared Language Handbook," *Science of Computer Programming*, 6, (April 1986), 134-157.

6. Guttag, J. V. and Horning, J. J. "A Specification of a Distributed Name Server," Internal Memorandum, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, 1986.

7. Guttag, J. V. and Horning, J. J. "The Topaz Threads Synchronization Procedures," Internal Memorandum, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, 1986.

8. Guttag, J. V., Horning, J. J. and Wing, J. M. "Larch in Five Easy Pieces," Digital Equipment Corporation Systems Research Center Technical Report No. 5, Palo Alto, CA, July 1985.

9. Guttag, J. V., Horning, J. J. and Wing, J. M. "An Overview of the Larch Family of Specification Languages," *IEEE Software*, (September 1985), 24-35.

10. Liskov, B. H. and Guttag, J. V. *Abstraction and Specification in Program Development*, Cambridge, MA, MIT Press, 1986.

11. Yelick, "A Generalized Approach to Equational Unification," MIT/LCS/TR-344, MIT Laboratory for Computer Science, Cambridge, MA, August 1985.

Thesis Completed

1. Yelick, K. A. "A Generalized Approach to Equational Unification," S. M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 1985.

Theses in Progress

1. Hinman, D. "On the Design of Larch Interface Languages," S. M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1986.
2. Zachary, J. L. "Integrating Logic Programming and Data Abstraction," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected January 1987.

Talks

1. Garland, S. J. "Program Correctness," Friends of 2^7, Wellesley, MA, January 1986.
2. Garland, S. J. "Advanced Placement Computer Science," (panel discussion), ACM Computer Science Conference, Cincinnati, OH, February, 1986.
3. Guttag, J. V. "Reve and Larch," Digital Equipment Corporation Systems Research Center, October 1985.
4. Guttag, J. V. "Software Reliability, Fault Tolerance, and Specifications," IFIP WG 10.4, January 1986.
5. Guttag, J. V. "An Overview of Larch"

   Tektronix Computer Research Laboratory, February 1986
   University of Washington, February 1986
   Xerox Palo Alto Research Center, February 1986
   Universite de Paris 5-6, April 1986
6. Guttag, J. V. "The Specification of Distributed Systems," Tektronix Computer Research Laboratory, February 1986.
7. Guttag, J. V. "Specifying the Topaz Synchronization Primitives," Universite de Paris-Sud, April 1986.

THEORY OF COMPUTATION

Academic Staff

B. Awerbuch
P. Elias
S. Goldwasser
L. Heath
F.T. Leighton
C. Leiserson
N. Lynch
A. Meyer
S. Micali
R. Rivest, Group Leader
D. Shmoys
M. Sipser

Associates

B. Chor
O. Goldreich
B. Trakhtenbrot

Graduate Students

W. Aiello
D. Barrington
B. Berger
B. Bloom
R. Boppana
V. Breazu-Tannen
D. Britsimas
T. Bui
J. Buss
T. Cormen
P. Feldman
R. Greenberg
A. Goldberg
S. Goldman
J. Hastad
R. Hirschfeld
A. Ishii
B. Kaliski
J. Kilian
S. Kipnis
P. Klein
R. Koch
B. Maggs
F.M. Maley
S. Malitz
S. Mentzer
M. Newman
J. Park
C. Phillips
S. Rao
M. Reinhold
P. Rogaway
A. Sherman
J. Siskind
R. Sloan
S.-M. Wu
L. Yedwab

Undergraduate Students

T. Heigham
J. Hinsdale
C. Kaklamanis
T. Leung
B. Rogoff

Support Staff

A. Benford
B. Hubbard
L. Melcher
I. Radzihovsky
R. Spenser

Visitors

P. Gacs
S. Homer
E. Lander
L. Levin
R. Kemmerer
Y. Moses
J. Reif
A. Shamir
M. Wand

1. RESEARCH OVERVIEW

The Theory Group continues to be vital and prolific. Principal research areas are:

- algorithms: combinatorial, geometric, graph-theoretic, number theoretic
- cryptology
- computational complexity
- distributed computation: algorithms and semantics
- randomness in computation
- semantics and logic of programs
- VLSI design theory.

Group members were responsible for over one hundred publications during the past year, as well as dozens of public lectures around the world. The reader may review the *individual reports* and the *bibliography and lecture list* that follows for further descriptions of the year's activities.

The following major research accomplishments merit highlighting:

1) Goldreich and Micali's proof that all problem's in NP have "zero-knowledge proofs". As a consequence, any cryptographic protocol that is correct with respect to a very weak adversary can be automatically transformed into an equivalent protocol correct in the most adversarial scenario [17].

2) Goldwasser and Kilian's efficient probabilistic algorithm for generating certified primes numbers [20].

3) Leighton's minimax grid-matching algorithm and its application to average case analysis of algorithms for bin packing, dynamic allocation and wafer-scale integration [34], [12].

4) Meyer *et al.*'s analysis of polymorphic types in programming and the negative consequences of the *'type' is a type* assumption [10], [36], [37].

2. FACULTY AND RESEARCH ASSOCIATES

Baruch Awerbuch

Awerbuch has been working on designing efficient and reliable distributed protocols. He studied issues related to complexity of different distributed network protocols such as synchronization and deadlock resolution, as well as some classic graph-theoretic problems, like Breadth-First-Search and Spanning Tree. He also worked on cryptographic protocols for implementing simultaneous broadcast networks.

Research on network synchronization has provided a simple and efficient methodology for developing simple and efficient protocols in an asynchronous network.

This was achieved by developing a communication protocol, referred to as synchronizer, which transforms an arbitrary synchronous protocol in such a way that the resulting protocol runs correctly on an asynchronous network. The synchronizer is, in effect, a compiler of high-level synchronous algorithms on a low-level asynchronous machine. The proposed synchronizer is proved to be optimal as far as its time and communication requirements are concerned. It also yields improvements for existing problems, i.e., distributed Breadth-First-Search and Maximum Flow.

The research on distributed Breadth-First-Search algorithms was conducted jointly with R. Gallager (MIT). This problem arises in relation with the efficient routing of messages towards a certain destination node. A number of algorithms have been discovered whose complexities are close to optimal. Another graph problem that was investigated was the problem of constructing (distributedly) a Spanning Tree in the network. An optimal algorithm for this problem was found.

Research on detection and resolution of deadlocks in a communication network was conducted jointly with Micali. Deadlock occurs if the amount of work that the network attempts to perform requires more resources than the network actually has. As a result, different network transactions get stuck in the middle, since no transaction can get enough resources to proceed to its completion. Efficient solution of this problem would result in improved performances of communication networks, and would prevent wasting precious resources like communication bandwidth and memory space. The main contribution of this research is a new protocol for deadlock resolution, which deals with the most general form of deadlocks. Its complexities in communication, time, and memory requirements are optimal.

Cryptographic protocols in unreliable networks were investigated jointly with Chor, Goldwasser and Micali. The main problem being studied was implementing simultaneous networks, where the abilities of potential adversaries are limited. Many fundamental protocols (i.e., coin flipping, contract signing, etc.) can be implemented quite easily in simultaneous networks. The protocol which was developed simulates simultaneous network by a non-simultaneous network, provided that the number of adversaries is not too big.

Awerbuch plans to continue research in distributed algorithms and communication protocols. He is mainly interested in efficient algorithms for practice-oriented problems. In particular, he plans to work on protocols for implementing atomic access to shared registers for asynchronous system of processes and on hierarchical routing schemes in networks. Another problem he plans to work on is the development of modular ways of describing distributed algorithms.

Benny Chor

Chor worked on various aspects of randomized agreement algorithms in unreliable distributed environment. He studied the implementation of simultaneous broadcast in Byzantine environment (joint work with Goldwasser, Micali, and Awerbuch). Together with Merritt (MIT) and Shmoys, he developed protocols for distributed consensus in constant expected time with improved fault tolerance in various synchronous and asynchronous failure-by-omission scenarios. Chor wrote a survey paper (jointly with C. Dwork (IBM-SJ)) on randomized algorithms for fault-tolerant distributed agreement.

Chor investigated VLSI implementations of exponentiations in finite fields of characteristic two, which is the computational bottleneck of a public-key cryptosystem that he designed in his thesis with Rivest. In addition he worked on improving schemes for extracting unbiased bits from weak sources of randomness. He also worked on better lower bounds for probabilistic communication complexity (continuing past work with O. Goldreich).

Peter Elias

Two simple-minded data-compression algorithms have been analyzed which permit immediate encoding of each message without delay or knowledge of the statistical characteristics of the message source [13]. Receiver and transmitter need only agree in advance on an indexing of the possible messages by the positive integers and on a sequence of (variable-length) binary codewords which represents those integers, no one of which is the beginning of another. In the algorithm simplest to implement, the transmitter encodes each message into the codeword which represents the total number of messages sent since it last occurred. The data structure required is simply an array which stores in index order the counter reading at the most recent occurrence of each message; an implementation could run very fast. A somewhat better scheme, with much more complex data structure requirements, encodes each message into the codeword which represents the number of distinct messages sent since its last occurrence. It has also been analyzed by Bentley, *et al.* If a good set of codewords is chosen to represent the integers then either scheme can do almost as well on any finite sequence of text as would an off-line Huffman encoding which measured the frequencies of the letters in the sequence and designed a code to match, providing that the information per letter of text is not too small: for example, if the text is taken to be a sequence of words rather than letters. Other adaptive on-line schemes can do better, but at added costs in complexity, both of concept and of necessary data structure.

Oded Goldreich

Goldreich has been mainly interested in two topics: zero-knowledge proofs and software protection. In addition he improved the Goldwasser-Micali-Rivest signature scheme, and have been working with Chor on developing better schemes for extracting unbiased bits from weak sources of randomness.

Together with S. Micali and A. Wigderson (UC-Berkeley), he has demonstrated the generality and wide applicability of zero-knowledge proofs, a fundamental notion introduced by Goldwasser, Micali and Rackoff (Toronto). Zero-knowledge proofs are probabilistic and interactive proofs that efficiently demonstrate membership in a language without conveying any additional knowledge. So far, zero-knowledge proofs have been known for some number theoretic languages in the intersection of NP and Co-NP. Under the assumption that encryption functions exist, it is shown that all languages in NP possess zero-knowledge proofs. This result is used to prove two fundamental theorems in the field of cryptographic protocols. These theorems consist of standard and efficient transformations that, given a protocol that is correct with respect to a very weak adversary, output a protocol correct in the most adversarial scenario. Using no assumptions, zero-knowledge proofs for both graph isomorphism and graph non-isomorphism, are presented.

Software protection is one of the most important issues concerning computer practice. The problem is to sell programs that can be executed by the buyer, yet cannot be duplicated and/or distributed by him to other users. Goldreich made the first steps towards a theoretic treatment of software protection, by distilling and formulating the key problem of learning about a program from its execution, and by presenting an efficient way of executing programs (i.e., an interpreter) such that it is infeasible to learn anything about the program by monitoring its executions. Current cryptographic techniques can be applied to keep the contents of the memory unknown throughout the execution, but are not applicable to the problem of hiding the access pattern. Hiding the access pattern efficiently is the essence of new solution. It is shown how to implement (on-line) t fetch instructions to a memory of size m by making tm^{ε} actual accesses, for every fixed $\varepsilon>0$. A reasonable scheme that protects against duplication follows.

Goldreich served on the program committee of Crypto85 conference.

Shafi Goldwasser

Goldwasser has done research in two areas: computational number theory and the study of interactive proof systems.

Goldwasser and Kilian devised a new probabilistic primality test which outputs a "short" (deterministic polynomial time verifiable) proof of correctness of its assertions of primality and compositeness. Thus its assertions of primality are certain rather than being correct with high probability (if a fair coin is used) or dependent on unproven assertions as in the tests of Miller, Solovay-Strassen, and Rabin in that its assertions of primality are certain, rather than being correct with high probability. The test terminates in expected polynomial time on all but at most an exponentially vanishing fraction of the inputs of every length.

This result shows that:

- There exist an infinite set of primes which can be recognized in expected polynomial time.
- Large certified primes can be generated in expected polynomial time (all previous methods were heuristics).

Under a very plausible condition on the distribution of primes in "small" intervals, the proposed algorithm can be shown to run in expected polynomial time on every input. This condition is implied by Cramer's conjecture. The methods employed are from the theory of elliptic curves over finite fields.

Goldwasser and Sipser examined the relative power of two interactive proof systems generalizing NP. An interactive proof system is a method by which one party of unlimited resources, the "prover", can convince a party of limited resources, the "verifier," of the truth of a proposition. The verifier may toss coins, ask repeated questions of the prover, and run efficient tests upon the prover's responses before deciding whether to be convinced. This extends the familiar proof system implicit in the notion of NP in that there the verifier may not toss coins or speak, but only listen and verify. Interactive proof systems may not yield proof in the strict mathematical sense: the "proofs" are probabilistic with an exponentially small, though non-zero chance of error.

Two notions of interactive proof system have been defined. One, by Goldwasser, Micali, and Rackoff permits the verifier a coin that can be tossed in "private" i.e., a secret source of randomness. The second, due to Babai requires that the outcome of the verifier's coin tosses be "public" and thus accessible to the prover.

Goldwasser and Sipser show that the two systems are equivalent in power with respect to language recognition. This result has implications on the ease of design of cryptographic protocols, as interactive proof systems provide a useful framework for studying the correctness of cryptographic protocols.

Goldwasser has co-organized an NSF sponsored workshop on the Mathematical Theory of Security, which was held in Endicott house in the past summer. She is currently on the program committee of Crypto 86.

Leonard S. Heath

Heath completed his PhD in computer science at the University of North Carolina in August, 1986 [26]. He has worked on the book embedding problem, which is related to the realization of arrays of VLSI processors fault-tolerantly. He has developed algorithms for embedding various classes of graphs in books [27], [28]. Recently, Heath and Istrail (Wesleyan University) have obtained an efficient algorithm for embedding a graph of genus g in a book of $O(g)$ pages [29]. This disproves an implicit conjecture of Bernhart and Kainen that the page number of genus g graphs is unbounded.

Heath will continue research into approaches for fault-tolerant computing. He hopes to develop approaches that are both theoretically sound and practical.

Tom Leighton

The highlight of the year is the joint work with P. Shor (MSRI) and E. Coffman (Bell Labs) on the minimax grid matching problem and its application to the average case analysis of algorithms for such problems as bin packing, dynamic allocation and wafer-scale integration. The work also turned out to have applications to some planar matching problems, possibly to testing pseudorandom number generators and to some problems in mathematical statistics. Leighton spoke about some of this material at last year's review, and is in the process of finishing off the related papers now.

Leighton is continuing work on channel routing with Berger, D. Brown (U. Colorado), and M. Brady (U. Illinois). They are in the process of improving the bounds and algorithms for channel routing in two or more layers. The results are not yet practical, but are definitely getting closer. The last "factor of two" from the channel widths produced by the algorithm has been eliminated, and the focus is now on the additive terms.

Leighton is also continuing work on the development of algorithms for integrating two-dimensional arrays on wafers that contain faults. He is looking at both worst case and average case models, and is optimistic that recent results will prove to be practical. Leighton is working with a UROP student over the summer to implement the most recent algorithms on realistic problems.

In the coming year, Leighton hopes to return to earlier work on the problem of efficiently routing messages in a fixed connection network, an increasingly important problem in parallel computation and an old favorite. The problems in this area have mostly been solved "theoretically" but most of the solutions are inefficient for small values of N (say $N \leq 2^{20}$), which of course is where all the action is. It would be nice to find a theoretically sound and practically efficient scheme for routing messages in a high bandwidth, low diameter network such as the hypercube.

Leighton is also planning to work on the design and analysis of ethernet protocols which work well on average. Currently, most ethernet systems use a protocol called exponential backoff to decide when to retransmit a message that got garbled because of simultaneous transmissions. While this has worked reasonably well in practice so far, there is very strong evidence that the protocol will start to fail dramatically as the traffic on typical ethernets continues to increase. He has an excellent candidate for replacing exponential backoff, and are currently trying to prove that it will work, even for very high levels of message traffic. The experimental evidence is very encouraging and already Hastad (who will postdoc in the Math Dept.. next year) and Leighton have proved some fairly positive lemmas concerning performance.

By far the biggest project (both recently and in the coming year) is Leighton's book on "An Introduction to the Theory of Networks, Parallel Computation, and VLSI Design." The book is based on the course that he has taught for the last three years on parallel computation and VLSI design, and is planned to be usable as a textbook and a reference text for people working in the field (probably a hopeless combination). So far, about 100 pages have been written.

Charles Leiserson

Leiserson has continued his work on volume-universal networks and now has several improved designs. One simple network contains only *n* small switches for a network with *n* processors and is a generalization of the Benes network. He and Greenberg have discovered good message-routing algorithms for this and other volume-universal networks. He has also developed with Maggs a PRAM-like abstraction of volume-universal networks. The new model, called a *distributed random-access machine* (DRAM) allows the communication requirements of an algorithm to be effectively measured.

Leiserson has also been working with Ishii on the semantics and timing analysis of clocked circuits, and with Kilian and Kipnis on understanding the power of multiple-pin interconnections. He was the program chairman for the recent Fourth MIT Conference on Advanced Research in VLSI, the proceedings of which is now a book from MIT Press. He also won the Best Presentation Award for his paper on fat-trees at the 1985 International Conference on Parallel Processing.

Nancy Lynch

Lynch's research is on theoretical aspects of distributed computation. This involves design and analysis of distributed algorithms, proofs of lower bounds, and formal modeling of distributed algorithms and systems.

A major effort was a system modeling project, with collaborator Dr. M. Merritt (AT&T Bell Labs), in the area of distributed database concurrency control and resiliency. This area is of critical importance to distributed computing, but the work was previously described in hundreds of unrelated research papers, with no common framework to aid in comprehension. Lynch and Merritt developed a satisfactory common framework, and used it to present and verify, an important exclusive-locking algorithm for concurrency control and resiliency. They are now extending the results to many other algorithms.

Other work involved specifying properties that can be guaranteed by (nonserializable) highly available replicated database systems. Another project, joint with student M. Tuttle, involved establishing a basic model for concurrent computation which could be used in carrying out hierarchical correctness proofs for distributed algorithms. Work with students B. Coan, J. Lundelius and A. Fekete involved determining the costs of solving various problems of reaching consensus in fault-prone distributed networks.

For further details and references, see the report of the Theory of Distributed Computation Research Group.

Albert R. Meyer

Meyer's research focuses on the semantics and logic of programming languages. He recently solved an open problem of some years' standing about axiomatic semantics of programs [38]. His joint paper with Reinhold [37] analyzes the differing criteria desired or expected of typechecking disciplines and establishes that the appealing 'type of all types' concept of polymorphic type discipline violates many of these criteria. Meyer also gave an invited lecture on these issues at the 1986 ACM Symposium on Principles of Programming Languages. For further information on specific research topics, see the references [40], [39], [10], [36], [9], and the personal reports of his students Breazu-Tannen and Reinhold (working on type theory), and Bloom (on concurrency).

Meyer supervised two weekly research seminars, one on semantics and logic of concurrent processes (jointly with Lynch) and another on types in programming (jointly with Trakhtenbrot). Both seminars were attended by faculty from several local universities as well as MIT. He was Program Chairman of the newly organized IEEE Symposium on Logic in Computer Science held June 16-18, 1986. He continues as Editor(-in-Chief) of the journal Information and Control, and is an Editor of a major Handbook of Theoretical Computer Science to be published in 1987 by North-Holland.

Silvio Micali

One focus of Micali's research has been on the use of randomness for proving theorems. Micali and coauthors have explored new avenues for efficiently proving theorems that would allow one to handle more theorems than in the traditional "NP" framework. So far, for example, it was not known whether a "prover", even having infinite power, could quickly convince a polynomial-time "verifier" that two given graphs are *not* isomorphic. These investigations resulted in a new, probabilistic and interactive way of proving theorems. For example, this way is powerful enough to allow an efficient and interactive verification of graph non-isomorphism. Micali intends to determine the power of the theorem-proving procedure and, more generally, to determine what theorems can be proved in any other efficient way.

Another focus is measuring and reducing the "amount of knowledge" conveyed by a communication. This is the basis of a rigorous and general study of the slippery field of cryptographic protocols. Of particular interest is a new "compiler-type" algorithm that seems fundamental in the ndesign of cryptographic protocols with more than two participants. This algorithm, given the specification of *any* protocol for totally honest parties (which is usually trivial to design), outputs an alternative and equivalent protocol that *remains correct* even if up to *half* of its participants deviate for their prescribed

programs in an *arbitrary* (but polynomial-time bounded) way. Micali proposes to extend his "compiler" to handle the case of two-party protocols as well.

He also proposes to develop theory in the field of asynchronous distributed systems when all participants are reliable. Here the adversary to defeat is the arbitrary arrival time of messages. The only guarantee we have is that all messages will all be eventually delivered. However, messages sent first may arrive later and we do not have an upper bound the time that will take for a message to reach its recipient. Clearly, finding robust good solutions in this extreme model will, *a fortiori*, imply having found good solutions for more reasonable models.

Ronald L. Rivest

Goldman and Rivest have been investigating ways to improve the efficiency of computing the maximum-entropy probability distribution, given a set of constraints (in terms of expected values) for the unknown distribution. They are working on improvements to a recent approach suggested by Peter Cheeseman (NASA), which is similar in spirit to the fill-in-minimization problem encountered when solving sparse linear systems. They expect that the techniques being developed will yield very substantial improvements in the running time, in practice. (Some experimentation is needed to validate their ideas on realistic examples since the running times are exponentially sensitive to certain characteristics of the input.)

Rivest has invented three new algorithms for the control of a broadcast channel (of the slotted-ALOHA type). These algorithms are called Bayesian Broadcast, Pseudo-Bayesian Broadcast, and Recursive Pseudo-Bayesian Broadcast. The essential characteristic of these algorithms is that each station will use Bayes' Rule to estimate the probability, for each r, that exactly r stations have a packet to transmit in the next time slot. Here each station receives feedback as to whether each slot was empty, successful, or contained a collision. Experimental results (conducted by Michelle Lee), show that the Pseudo-Bayesian Broadcast algorithm has significantly less delay than previously published algorithms. It is also extremely simple to implement. (More recently, J. Tsitsiklis (MIT) has proven that it is stable with a packet arrival rate of up to e^{-1} packets/slot.)

Rivest has worked with Bloom on "Abstract AI" -- the study of computational mechanisms attempting to learn an environment into which they have been placed. They derived conditions on the environment which permit the environment to be learned effectively.

With Sherman and Kaliski, Rivest has investigated the security of the Data Encryption Standard (DES). Using a special-purpose board for computing DES quickly on an IBM PC, they ran experiments to determine whether DES suffers from being a pure cipher.

(Such a weakness would allow a very efficient attack to be mounted against DES.) They found no such weakness.

David Shmoys

Shmoys has been studying several problems in fault-tolerant distributed computing. One of the fundamental problems in this area is "Byzantine Agreement", where each processor starts with a private input and at termination all processors that operate correctly agree on an output that depends non-trivially on the input (i.e., if all correct processors have the same input bit *b*, then the output bit is *b*.) Shmoys, in joint work with Chor and M. Merritt (MIT), gave a family of protocols that terminate in constant expected time while tolerating a linear fraction of the processors failing. The results hold in a variety of computation models, both synchronous and asynchronous. However, this algorithm tolerates only a benign type of failure, where messages may not be delivered, but messages which arrive are correct. In separate work, Shmoys also provided a family of protocols that could handle arbitrarily malicious behavior by faulty processors, but for this approach to tolerate a linear fraction of faulty processors, the expected completion time grows to log log *n*.

Shmoys also continued his investigations in the design of provably good heuristic procedures for the solution of intractable optimization problems. The central notion in this work is that of a dual approximation algorithm. A traditional (or primal) approximation algorithm seeks a feasible but suboptimal solution for the problem where the degree of suboptimality possible is bounded. A dual approximation algorithm seeks a superoptimal but infeasible solution for the problem where the degree of infeasibility possible is bounded. In addition to their immediate application, Shmoys has shown that dual approximation algorithms are often useful in the design of primal approximation algorithms. This general strategy was applied in the case of several scheduling problems. For all of these problems the aim is to assign a given set of jobs to a set of parallel processors, so that all jobs are completed as quickly as possible. If the machines run at identical speeds, or if the processing requirement for each job varies uniformly on different machines, Shmoys proved that there is a polynomial approximation scheme, thereby showing that solutions arbitrarily close to the optimum can be obtained. In addition, Shmoys showed that if the processing times varied arbitrarily from machine to machine, then there was an algorithm that produced solutions within a logarithmic factor of the optimum.

Michael Sipser

Sipser has been continuing his work on questions in complexity theory. Recent results include:

- the equivalence of public-coin and private-coin interactive proof systems joint with Goldwasser [19] described in her section above,

- a connection between expander graphs and the relative power of time versus space [43]. This second result shows that explicit construction of certain kinds of expander graphs would enable their use as "randomness magnifiers" for simulating randomized computations.

Current emphasis is on developing new methods for proving the inherent computational complexity of a variety of problems in different settings. This Fall, M. Sipser will conduct a seminar to survey the latest results and to discuss further directions.

B.A. Trakhtenbrot

Trakhtenbrot's research focus is on computability, sequentiality, and invariance of functionals on the one hand, and the language constructs which are able to express them on the other hand. The goal is to develop a unifying approach based on the theory of typed lambda-calculus and logical relations. Current plans are about the comparative study of different approaches to sequentiality:

- the semantical approach (as in earlier work with his students);
- the syntactical approach (as in the rewriting paradigm);
- interpretations used in the theory of concurrency.

3. STUDENTS AND VISITORS

William Aiello

Aiello has been working with Goldwasser and Hastad on open problems concerning Arthur-Merlin games and Interactive Proofs (IP). Recently, they constructed an oracle which separates the class of languages recognized by IP from the polynomial hierarchy. In particular, this implies that relative to our oracle more languages can be recognized by IP's (or Arthur-Merlin games) with a polynomial number of interactions than with a constant number. This in turn gives evidence that the unrelativized statement is also true. We also hope to show that Babai's methods for proving that the AM hierarchy collapses at the second level are optimal.

Dave Barrington

Barrington has been finishing up his Ph.D. research under Sipser in the Math department. He has been studying branching programs, a model of computation where the settings of Boolean input variables determine a path through a directed acyclic graph to a final node which accepts or rejects the input. He showed that branching programs of constant width and polynomial size are much more powerful than previously believed --- they can simulate all of the parallel complexity class NC^1 (Boolean circuits of fan-in two and depth $O(\log n)$.

Further work has shown that this unexpected power comes from the ability of branching programs of width at least five to represent non-solvable permutation groups. Recently, with Denis Therien of McGill University, Barrington has used the connection between branching programs and finite monoids to give new characterizations of important subclasses of NC^1. He has also applied his main result to show that Boolean circuits of width four and polynomial size can recognize exactly NC^1.

Barrington has been working with complexity classes with very constrained resources, such as NC^1 or log-space Turing machines. These classes can serve as an important laboratory for developing insights about more general computations and the big questions --- what do non-determinism, randomness, parallelism, or non-uniformity add to computational power?

In Fall, '86, he will take a faculty position in the Computer and Information Sciences department of the University of Massachusetts, Amherst, but expects to be a frequent visitor to MIT.

Bonnie A. Berger

Berger and Leighton have worked on achieving better bounds and algorithms for channel routing. Channel routing plays a crucial role in the development of automated layout systems for integrated circuits. Jointly with Brady (U. Ill.) and Brown (U. Col.), they have developed algorithms for routing channels with several layers [3]. They not only have achieved substantial new results, but have provided a unified framework in which many previously known results can be obtained.

For the unit-vertical-overlap model, they have developed a 2-layer channel routing algorithm which uses at most $d+O(d^{1/2})$ tracks to route two-point net problems and $2d+O(d^{1/2})$ tracks to route multipoint nets. They have also shown a $d+\Omega(\log d)$ lower bound for routing two-point nets on 2 layers even when unrestricted vertical overlap is allowed; hence, their upper bound is nearly optimal even in a more general setting. Moreover, they have demonstrated the robustness of their algorithm by showing how it can be used to obtain the known bounds for the Manhattan and knock-knee models.

Finally, they generalized the algorithm to unrestricted multilayer routing and used only $d/(L-1) + O(d^{1/2})$ tracks for two-point nets (within $O(d^{1/2})$ tracks of optimal) and $d/(L-2) + O(d^{1/2})$ tracks for multipoint net problems (within a factor of $(L-1)/(L-2)$ times optimal).

Bard Bloom

Meyer and Bloom are studying the denotational semantics of parallel and nondeterministic processes. Dana Scott's very successful models for the semantics of sequential, deterministic programs do not extend naturally to the more general domain. There are a number of proposals for a replacement; Meyer and Bloom are investigating several of these models.

Meyer and Bloom are also investigating the complexity of the operational semantics of Milner's SCCS. They have proved that the problem of deciding if an arbitrary program can take a single step is not recursively enumerable, under some barely reasonable assumptions about the set of actions. They are in the process of extending this result to completely reasonable assumptions.

Ravi B. Boppana

Boppana has worked on proving lower bounds for several restricted models of Boolean circuits. His Ph.D. thesis [8] on this topic was completed in May 1986.

The first circuit model studied is monotone circuits. A major development appeared in [A. A. Razborov, "Lower bounds for the monotone complexity of some Boolean functions," Dokl. Ak. Nauk. SSSR 281 (1985), pp. 798-801 (in Russian). English translation in Sov. Math. Dokl. 31 (1985), pp. 354-357] which showed that monotone circuits detecting cliques in graphs must have superpolynomial size. Alon and Boppana [2] show that the lower bound can be improved to exponential size.

The second model is that of Boolean formulas. [L.G. Valiant, "Short monotone formulae for the majority function," J. Algorithms 5, 363-366, 1984] shows that the majority function has short monotone formulas; his construction uses the general technique of combining independent copies of probabilistic formulas to amplify the performance of the formulas. Boppana [4] showed that the amount of amplification obtained by Valiant is actually best possible.

Lagarias and Boppana [[illegible]] studied the computational complexity of checking a relation versus evaluating a relation. Hirschfeld and Boppana [6] wrote an expository paper on pseudorandom number generators and their relationship to complexity theory. Hastad and Boppana [5] studied approximation properties of constant depth circuits.

Boppana plans to continue work in Boolean circuit complexity. In particular, he will investigate whether the recent lower bounds for monotone circuits can be extended to the nonmonotone case.

Val Breazu-Tannen

Breazu-Tannen has studied equational type disciplines (see [9]) and λ-calculus with least fixpoint operators. He served as organizer for the TOC " Types in Programming" seminar where he also made several presentations.

His current interests include:

- Conservative extension theorems in typed λ-calculi; Polymorphism and second-order λ-calculus; Dependent types and the calculus of constructions; Relations between type disciplines.
- λ-calculus with equational type disciplines (e.g., types satisfying domain equations); Type coercion.

- λ-calculus with least fixpoint operators.
- Type theory; Higher-order categorical logic (cartesian closed categories, topoi); Higher-order algebraic theories.
- Logical relations and representation independence theorems.
- Learning about *concrete* programming environments to which the ideas of the above fields of theoretical investigation apply.

Breazu-Tannen's current research focuses on conservative extension theorems about programming language data types. The history of programming language design has seen a progressive addition of powerful features such as higher-order types, recursively defined types, polymorphic types, dependent types and others, giving rise to more and more complex type disciplines. Such features are theoretically modeled by various *typed λ-calculi.* Adding a new feature corresponds to seeing the new calculus as an *extension* of an old one. If the old calculus is consistent, is the new one too? A stronger question is: are all old program phrases that were not provably equivalent in the old calculus still not provably equivalent in the extended calculus? (i.e., is the extension conservative?) An affirmative answer provides formal evidence for the *orthogonality* of the new feature with respect to the the old ones, a well-established desideratum in programming language design. Both model-theoretic and proof-theoretic techniques have been used to establish conservative extension, e.g., finitely typed λ-calculus is conservative over any first-order equational theory, but untyped λ-calculus is not.

Jonathan Buss

Buss' research has been in the area of relativized complexity theory. Previous work in this area has been largely concerned with polynomial-time-bounded machines. In the case of space-bounded machines, there are several possible models in the literature, none of which is entirely satisfactory. In particular, results connecting time and space do not relativize in the previous models. These definitional problems have obstructed progress in this area.

Principal work has been the formulation of a new model for alternation oracle machines. In the new model, the connections between time and space bounds provided by alternation (ALOGSPACE=P, AP=PSPACE) hold relative to any oracle. When the model is reduced to the case of deterministic machines, a slightly different model of relativized time and a new model of relativized space result. These deterministic models are better-behaved than the standard models. Several results on the simulation of time by space and the simulation of multihead Turing machines hold for all oracles only in the new model. This work is reported in his doctoral thesis [11].

Continuing work is concerned with the implication of the new model for log-space hierarchies and with relativized probabilistic machines.

Tom Cormen

Cormen has continued his work in VLSI designs of concentrator switches, building on earlier joint work with Leiserson in designing a fast hyperconcentrator switch. Addressing the problem of pin boundedness, Cormen has designed an (N,m,α)-partial concentrator switch based on a recent technique for sorting on a mesh. This switch uses the earlier hyperconcentrator switch as a subcircuit, requires at most $2N^{1/2} + 1/2 \log N$ data pins per chip, has $\alpha = 1\text{-}O(N^{3/4}/m)$, and can be packaged in three dimensions with volume $O(N^{3/2})$. The switch can route any set of $k \leq \alpha m$ bit-serial messages from input wires to output wires. A signal incurs at most $7/2 \log N + O(1)$ gate delays in passing through the switch.

Paul Feldman

Feldman has been working with Micali on cryptographic protocols even before transferring to MIT from Harvard this year. One result of this collaboration was an algorithm for reaching Byzantine agreement in constant expected time whenever fewer than a third of the processors are faulty [14]. Another result, an efficient scheme for verifiable secret sharing, is expected to be included as part of his doctoral thesis.

Feldman obtained some results about connecting networks [15] working with Joel Friedman and Nick Pippinger at IBM-San Jose.

Feldman has been interested in the field of interactive proof systems, and has shown that if any prover can convince a polynomial time verifier about membership in a certain language, then there exists a polynomial space prover that can do the same.

Andrew V. Goldberg

Goldberg has been working on network flow problems. He coauthored a paper [18] that introduces a new method for solving a classical max-flow problem. Currently he is looking at other network flow problems, including 0-1 flows, multicommodity flows, load balancing, etc. His other interest include parallel algorithms, parallel architectures, and complexity theory.

Sally A. Goldman

Goldman has been working with Rivest on theoretical aspects of artificial intelligence. Specifically, they are studying the problem of calculating the maximum entropy distribution satisfying a given set of constraints. These constraints may be given by an expert or discovered by a learning program. They have compared several methods from the literature as well as beginning work on developing a new method. Their method simplifies maximum entropy computations by adding extra constraints.

Ron Greenberg

Greenberg has continued his investigation of the fat-tree architecture for general-

purpose parallel supercomputing. The randomized routing algorithm for fat-trees which he developed with Leiserson demonstrates that fat-trees are universal routing networks [21]. That is, in a VLSI model equating hardware cost with physical volume, any routing network can be efficiently simulated by a fat-tree of comparable hardware cost.

Johan Hastad

The two major topics of Hastad's research have been integer lattices and lower bounds on circuit complexity.

The work on lattices consists of two different types of work. First applying old techniques to new problems and secondly to find new techniques. In [22] the LLL algorithm is applied to prove that encoding linearly related messages with RSA with low exponent is insecure. In the area of new techniques the joint paper [24] proposes a polynomial time algorithm which finds integer relations among real numbers. The algorithm works in the model where real arithmetic can be done at unit cost.

The work on lower bounds for circuits has been concentrated on small depth circuits. Hastad has established almost optimal lower bounds for small depth circuits computing functions like parity and majority [23]. He has also proved that for any constant k there are functions which have linear size circuits of depth k but which require exponential size circuits for depth k-1. This work is included in his recently completed Ph.D. thesis [25].

These lower bounds have implications for relativized complexity. With Aiello and Goldwasser [1], Hastad has used similar methods to prove that there is an oracle A and a language L(A) with the following property: L(A) is not in the polynomial time hierarchy relative to A, but given the oracle A it is possible to effectively prove membership in L(A) using a randomized interactive proof.

Future plans include trying to prove lower bounds for circuits with less severe restrictions on the depth. For instance to prove that some explicit function is not in NC^1. He would also like to investigate further the notion of interactive proof.

Rafael Hirschfeld

The topics of Hirschfeld's research are pseudorandom number generators and complexity theory. He has investigated necessary and sufficient conditions for the existence of pseudorandom generators that cannot feasibly be distinguished from true random sources, as well as the implications of the existence of such generators for the relationship between deterministic and probabilistic computation. Boppana and Hirschfeld [6] have written an expository paper on this and related work.

Alexander T. Ishii

Ishii, in joint work with Leiserson, has been studying the analysis of clocked multi-

phase VLSI systems. Progress to date includes the development of a new semantic model for the functionality of VLSI systems. Promising aspects of the model include strong correspondence to widely held intuitive notions of functionality, and freedom from *a priori* assumptions about the relationship between different clock phases. In addition, study of the model has resulted in the identification of a number of clocking phenomena, which raise questions about the ability of standard simulation and analysis techniques to verify an optimally clocked circuit. It is hoped that continued study will result in the formulation of an algorithm to compute provably optimal clock phases for any specific VLSI system.

Burt Kaliski

Kaliski completed experimental research on the Data Encryption Standard (DES) and studied applications of elliptic curves to cryptography.

The DES research, with Rivest and Sherman, involved the design, implementation and testing of special-purpose hardware for an IBM PC to perform the 2^{32} or more DES encryptions per day necessary for a variety of statistical tests. In summer 1985, the board was used to perform eight experiments, most of which confirmed expected properties of DES (i.e., that it behaves like a set of randomly-chosen permutations). The experiments also uncovered fixed points for the "weak keys" in DES.

Some of the most interesting recent developments in computational number theory are related to elliptic curves. H.W. Lenstra's algorithm for integer factorization, and two other related algorithms (by Rene Schoof and Victor Miller) all were published in the last two years, following a century of theoretical development. The elliptic curve study, with Rivest and Micali, and helpful insight from Goldreich, sought to apply the *elliptic logarithm problem* to cryptography, just as several other hard problems (factoring, discrete logarithm) have been applied. The main result is the construction of a *pseudo-random bit generator* using elliptic curves. Two related results of independent interest were also developed: a method of determining the order of an element in an arbitrary abelian group, with negligible error; and an oracle proof method for the simultaneous security of multiple bits of a discrete logarithm in an arbitrary abelian group.

Future work will include the documentation of the special-purpose DES hardware, and extensions of the results on elliptic curves. The latter may involve generalization to more complicated algebraic structures, an attempt at a sub-exponential time algorithm for computing elliptic logarithms, or the use of elliptic curves in "certifying" elements that generate a cyclic group.

Joe Kilian

Kilian's chief work has been a joint paper with Goldwasser [20] in which they apply the theory of elliptic curves to solve some open problems relating to primality testing.

He is also working on a paper with Kipnis and Leiserson applying elementary group theory to some problems concerning the power of multi-point nets.

Philip Klein

Klein has been studying parallel algorithms for language-theoretic and graph-theoretic problems. He has revised his joint paper with Reif [30] on a parallel algorithm for recognition of any fixed deterministic context-free language. Previously a parallel algorithm existed for general context-free language recognition, but this algorithm was slower and used many more processors.

For his Masters' thesis, Klein has developed, in collaboration with Reif, a new parallel algorithm for finding a planar embedding of a graph [31].

Klein will continue to work towards developing new efficient parallel algorithms.

Shlomo Kipnis

During this first year at MIT, Kipnis worked with Leiserson on theoretical VLSI problems. They tried to investigate multiple-pin nets, where each net may connect more than two processors (chips). Attention was given mainly to networks that realize some group of permutations among the chips. Together with Kilian some theoretical results from group theory were applied that characterize some networks.

Currently he is gathering the results on this topic into a paper with Leiserson and Kilian. He intends to continue research in this direction and related VLSI problems.

Bruce Maggs

Leiserson and Maggs have shown that many graph problems can be solved in parallel, not only with polylogarithmic performance, but with efficient communication at each step of the computation. The communication requirements of an algorithm are measured in a parallel random-access machine model called the distributed random-access machine, which can be viewed as an abstraction of volume- and area-universal networks such as fat-trees. In this model, communication cost is measured in terms of the congestion across cuts of the machine. The graph algorithms are based on a generalization of the prefix computation on lists to trees. These treefix computations, which can be performed in a communication-efficient fashion using a variant of the tree contraction technique of Miller and Reif, simplify many parallel graph algorithms in the literature.

Miller Maley

Maley continued his work on the mathematical foundations of wire routing. Using ideas from algebraic topology, he began developing a theory of planar wiring under homotopy constraints. This research aims to extend the scope of proposed algorithms for single-layer routing and compaction of VLSI layouts. The new theory facilitates the construction and rigorous justification of such algorithms.

Seth Malitz

Malitz has established some surprising facts connecting measure- and graph-theoretic properties of infinite graphs whose edges are represented by points on the real unit square [35]. Future plans are to work with Goldwasser and Sipser on interactive proof systems and with Bjorner (MIT Math.) on continuous analogues of finite lattices.

Yoram Moses

See the report of the Theory of Distributed Systems Group.

Cynthia Phillips

Phillips completed her Master's Thesis on space-efficient algorithms for computational geometry [41] supervised by Leiserson. She is currently investigating several possible topics for her Ph.D. research including parallel data structures, theory of VLSI and parallel computation, and pseudorandom permuter circuits.

Satish Rao

Rao is continuing work on a Master's thesis (begun at Bell Laboratories) on finding optimal separators for planar graphs. He currently is working with some success on restricted versions of the problem, but hopes to eventually tackle the more general problem.

Mark B. Reinhold

Dependent function types, which originated in the type theory of intuitionistic mathematics, have recently appeared in programming languages such as CLU, Pebble, and Russell. (A function has a dependent type when the type of its result depends upon the value of its argument.)

In [37], Reinhold and Meyer investigate the consequences of assuming that there exists a *type of all types* in a λ-calculus with dependent function types. The type of all types is the type of every type, including itself. When a language with dependent function types is enriched with the type-of-all-types assumption, enormous expressive power is gained at very little apparent cost. By reconstructing and analyzing a paradox of Girard, they show that this combination leads to several serious problems. The most significant of these are that for such a language (1) typechecking is undecidable, and (2) classical reasoning about programs is not sound.

The technical properties if λ-calculus with dependent types are complex, and Reinhold is currently developing the basic theory of conversion and type-checking for such languages in his Master's thesis.

Alan Sherman

Sherman, with Kaliski and Rivest, has been investigating the relationship between

algebraic and security properties of cryptosystems [33], [32], [42]. Using special-purpose hardware, they carried out a series of cycling experiments on the Data Encryption Standard (DES) to see if DES has certain extreme algebraic weaknesses. Their experiments show, with overwhelming confidence, that the set of DES transformations does not form a group under functional composition. Their experiments also detected fixed points for the so-called "weak-key" transformations, thereby revealing a previously unpublished additional weakness of the weak keys.

Jeffrey Mark Siskind

Siskind's research interests have centered around three different topics, namely silicon compilation from first principles, generalized phrase structure grammars, and logic programming and constraint propagation.

Silicon Compilation: All present silicon compilers, including MacPitts, generate layouts by using a set of predefined module generators. There are two fundamental problems with module generators. First, they are capable of producing only fixed classes of architectures. Second, they are technology dependent and require a significant effort to retarget to new technologies. The new approach, like MacPitts, separates the silicon compilation task into two phases, a translation from behavior to structure, and then a translation from structure to layout. Unlike MacPitts however, the intermediate structural format is a simple flattened network of transistors, rather than specifications for specialized module generators. Experimentation is underway with two different approaches towards the translation of this transistor network to efficient layout. The first uses a divide and conquer min-cut placement algorithm followed by a combination of Lee routing and slice and expand routing. The second more sophisticated approach is based on planar graphs. This work is being pursued under the supervision of Leiserson.

GPSG: Generalized Phrase Structure Grammars (GPSG) are a linguistic formalism developed by Gerald Gazdar for describing natural languages. They consist of small base set of context free grammar rules which are expanded by application of rule schemas, metarule transformations, and feature agreement to form a much larger set of context free rules which can be used to parse sentences. Several approaches are being explored towards the goal of testing the accuracy and adequacy of this formalism for both describing and parsing English. A GPSG grammar of about 100 rules has been constructed which can handle a fairly large subset of English. A system which expands this GPSG representation into a CFG representation has been constructed, and produces around 6000 CFG rules for the above GPSG rule set. The CFG rules are fed into an implementation of the Earley parsing algorithm and run against a small corpus of test sentences. This approach has proven to be very unwieldy and has prompted the construction of two alternative systems. The first is a modification of the Earley parser

to handle feature agreement directly by using a unification algorithm internally avoiding the need to expand the rule set. This alone is not sufficient as capability to handle derived rules without expansion must be provided. The second approach is to use Prolog based definite clause grammars which imply a depth first backtracking approach rather than the breadth first approach of Earley's algorithm. This work is being pursued under the supervision of Robert Berwick (MIT) and William Woods (Applied Expert Systems, Cambridge, MA).

Logic Programming: Two different paths of research in logic programming have been pursued until now. One path, centered around Prolog, provides unification and chronological backtracking as its main tenants. The second path, that of constraint propagation, focuses on dependency directed backtracking, but does not provide the necessary data and program abstraction capabilities to form a useful programming language. This research centers on a new language, Conlog, which merges the concept of unification (for providing data and program abstraction) with constraint propagation and dependency directed backtracking. As the new language is purely functional, simple syntactic transformations allow Conlog to express the lambda calculus as well, with the added benefit that many functions thus described are reversible. There are two new features which the above framework supports. The first is an extension of the dependency directed backtracking component to support generalized search of the solution space for minima, maxima, etc., in addition to just finding solutions. The second is a technique which is called intensional mode vs. extensional mode predicates, which allow a way of delaying and trading off computation. As a final benefit, viewing logic programming in the light of constraint propagation allows an implementation of Prolog for fine grain massively parallel computer architectures.

Bob Sloan

Sloan's major research activity has been to analyze several definitions of security for probabilistic public-key cryptosystems. He has been working with Micali, and has proved that some apparently distinct definitions are equivalent. His Master's thesis on this topic is in preparation.

Mitch Wand

Wand has recently obtained results on the completeness of type inference systems, extending the line of work published in the last three POPL conferences. These results concern the completeness of type inference systems, which infer type declarations from programs without them. Wand was able to formalize the proof of completeness of the basic system in a way which allowed easy extensions to more complex type systems than had previously been considered. From this work, the following papers are in progress:

- A Simple Algorithm and Proof for Type Inference
- Deriving Typechecking Rules for Macros
- A Complete Type Inference System for Simple Objects with Inheritance

Su-Ming Wu

For his Master's thesis, Wu will work with Micali on algorithms for finding maximum matchings in general graphs.

References

1. Aiello, W., Goldwasser, S. and Hastad, J. "On the Power of Interaction," *27th IEEE Symposium Foundations of Computer Science*, 1986, 368-379.
2. Alon, N. and Boppana, R.B. "The Monotone Circuit Complexity of Boolean Functions," *Combinatorica*, 1986, to appear.
3. Berger, B., Brady, M., Brown, D. and Leighton, F.T. "An Almost Optimal Algorithm for Multilayer Channel Routing," *27th IEEE Symposium Foundations of Computer Science*, 1986, submitted.
4. Boppana, R.B. "Amplification of Probabilistic Boolean Formulas," *26th IEEE Symposium Foundations of Computer Science*, 1985, 20-29.
5. Boppana, R. B. and Hastad, J. "Approximation Properties of Constant Depth Circuits," 1986, in preparation.
6. Boppana, R.B. and Hirschfeld, R. "Pseudorandom Generators and Complexity Classes," *Advances in Computing Research*, S. Micali (ed.), 1986, submitted.
7. Boppana, R.B. and Lagarias, J.C. "One-way Functions and Circuit Complexity," *ACM Structure in Complexity Theory Conference*, LNCS: 223, Springer-Verlag, 1986, 51-65.
8. Boppana, R.B. "Lower Bounds for Monotone Circuits and Formulas," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1986.
9. Breazu-Tannen, V. and Meyer, A.R. "Lambda Calculus with Constrained Types (Extended abstract)," *Logics of Programs*, LNCS: 193, R. Parikh (ed.), Springer-Verlag, 1985, 23-40.
10. Bruce, K., Meyer, A.R. and Mitchell, J.C. "The Semantics of Second-order Polymorphic Lambda Calculus," *Information and Control*, to appear, 1986. An earlier version authored by Bruce and Meyer appeared in Springer Lecture Notes in Computer Science, vol. 173.
11. Buss, J. "Relativized Alternation and Space-Bounded Computation," MIT Department of Mathematics, Cambridge, MA, 1986, to appear.
12. Coffman, E. and Leighton, F.T. "A Provably Efficient Algorithm for Dynamic Storage Allocation," *18th ACM Symposium Theory of Computing*, 1986, 77-90.
13. Elias, P. "Interval and Recency-rank Source Coding: Two On-line Adaptive Variable-length Schemes," *IEEE IT*, 1986. Also, to appear as MIT/LCS/TM-301.
14. Feldman, P. and Micali, S. "Byzantine Agreement in Constant Expected Time (and Trusting No One)," *26th IEEE Symposium Foundations of Computer Science*, 1985, 267-276.

15. Feldman, P., Friedman, J. and Pippinger, N. "Non-blocking Networks," *18th ACM Symposium Theory of Computing*, 1986, 247-253.

16. Goldreich, O., Micali, S. and Wigderson, A. "Proofs That Yield Nothing But Their Validity and a Methodology for Cryptographic Protocol Design," *27th IEEE Symposium Foundations of Computer Science*, 1986, 174-187.

17. Goldreich, O., Micali, S. and Wigderson, A. "Proofs That Yield Nothing But Their Validity, or All Languages in NP Have Zero-knowledge Proofs," MIT Laboratory for Computer Science and Yale University, 1986.

18. Goldberg, A.V. and Tarjan, R.E. "A New Approach to the Maximum Flow Problem," *18th ACM Symposium Theory of Computing*, 1986,136-146.

19. Goldwasser, S. and Sipser, M. "Interactive Proof Systems: Public vs. Private Coins," *18th ACM Symposium Theory of Computation*, 1986, 59-68.

20. Goldwasser, S. and Kilian, J. "Almost All Primes Can Be Quickly Certified," *18th ACM Symposium Theory of Computation*, 1986, 316-329.

21. Greenberg, R.I. and Leiserson, C.E. "Randomized Routing on Fat-trees," *26th IEEE Symposium Foundations of Computer Science*, 1985, 241-249

22. Hastad, J. "On Using RSA with Low Exponent in a Public Key Network," *SICOMP*, 1986, to appear.

23. Hastad, J. "Almost Optimal Lower Bounds for Small Depth Circuits," *18th ACM Symposium Theory of Computing*, 1986, 6-20.

24. Hastad, J., Just, B., Lagarias, J.C. and Schnorr, C.P. "On Finding Integer Relations Among Real Numbers," *SICOMP*, 1986, submitted.

25. Hastad, J. "Computational Limitations for Small Depth Circuits," Ph.D. dissertation, MIT Department of Mathematics, Cambridge, MA, 1986

26. Heath, L.S. "Algorithms for Embedding Graphs in Books," TR-85-028, University of North Carolina, Chapel Hill, NC, August 1985.

27. Heath, L.S. "Embedding Outerplanar Graphs in Small Books," *SIAM Journal Algebraic and Discrete Methods*, 1985, to appear.

28. Heath, L.S. "Embedding Trivalent Planar Graphs in Two Pages," *SICOMP* 1985, submitted.

29. Heath, L.S. and Istrail, S. "Surface-embeddable Graphs Can Be Embedded in a Bounded Number of Pages," *27th IEEE Symposium Foundations of Computer Science*, 1986, submitted.

30. Klein, P. and Reif, J. "Parallel Time O(log *n*) Acceptance of Deterministic CFL's," *SICOMP*, 1985, submitted. Also available as TR-05-84, Center for Research in Computing Technology, Harvard University.

31. Klein, P. and Reif, J. "An Efficient Parallel Algorithm for Planarity," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1986, in preparation.

32. Kaliski, B.S., Rivest, R.L. and Sherman, A.T. "Is DES a Pure Cipher? (Results of More Cycling Experiments on DES)," *Proceedings of Crypto 85*, Springer-Verlag, 1985, 212-222.

33. Kaliski, B.S., Rivest, R.L. and Sherman, A.T. "Is the Data Encryption Standard a Group?" *Advances in Cryptology: Eurocrypt 85*, Springer-Verlag, F. Pichler (ed.) 1986, 81-95.

34. Leighton, F.T. and Shor, P. "Tight Bounds for Minimax Grid Matching, With Applications to the Average Case Analysis of Algorithms," *18th ACM Symposium Theory of Computing*, 1986, 91-103.

35. Malitz, S.M. "Measures of Graphs on the Reals," 1986, to appear.

36. Mitchell, J.C. and Meyer, A.R. "Second-order Logical Relations (Extended Abstract)," *Logics of Programs*, LNCS: 193, R. Parikh (ed.), Springer-Verlag, 1985, 225-236.

37. Meyer, A.R. and Reinhold, M.B. "'Type' is Not a Type: Preliminary Report," *13th ACM Symposium Principles of Programming Languages*, 1986, 287-295.

38. Meyer, A.R. "Floyd-Hoare Logic Determines Semantics," *IEEE Symposium Logic in Computer Science*, 1986, 44-48.

39. Meyer, A.R. "Thirteen Puzzles in Programming Logic," *Proceedings DDC Workshop on Formal Software Development: Combining Specification Methods*, LNCS, Nyborg, Denmark, May 1984, to appear.

40. Parikh, R., Chandra, A., Halpern, J.Y. and Meyer, A.R. "Equations Between Regular Terms and An Application to Process Logic," *SIAM Journal of Computing*, 14, 4, (1985), 935-94.

41. Phillips, C.A. "Space-Efficient Algorithms for Computational Geometry," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1985.

42. Sherman, A.T. "Cryptology and VLSI (a two-part dissertation): I. Detecting and Exploiting Algebraic Weaknesses in Cryptosystems II. Algorithms for Placing Modules on a VLSI Chip," MIT/LCS/TR-381, MIT Laboratory for Computer Science, Cambridge, MA, 1986.

43. Sipser, M. "Expanders, Randomness, and Time versus Space," *Conference in Structure in Complexity Theory*, 1986, 325-329.

Publications

1. Aiello, W., Goldwasser, S. and Hastad, J. "On the Power of Interaction," *Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science*, 1986.

2. Alon, N. and Boppana, R.B. "The Monotone Circuit Complexity of Boolean Functions," *Combinatorica*, (1986), to appear.

3. Awerbuch, B. "A New Distributed Depth-first Search Algorithm," *Information Processing Letters*, 20, (1985), 147-150.

4. Awerbuch, B. "Complexity of Network Synchronization," *Journal of the ACM*, 32, (1985), 804-823.

5. Awerbuch, B. "Reducing Complexities of Distributed Maximum Flow and Breadth-first Search Algorithms by Means of Network Synchronization," *Networks*, 15, (1985), 425-437.

6. Awerbuch, B. "Communication-time Trade-offs in Network Synchronization," *4th ACM Symposium Principles of Distributed Computing*, R.H. Strong (ed.), 1985, 272-276.

7. Awerbuch, B. "Optimal Distributed Spanning Tree Algorithm," unpublished.

8. Awerbuch, B. and Gallager, R. "Distributed Breadth-first Search Algorithms," *26th IEEE Symposium Foundations of Computer Science*, 1985, 250-256.

9. Awerbuch, B. and Gallager, R. "Communication Complexity of Distributed Shortest Path Algorithms," LIDS-P-1473, Technical Report, MIT, Cambridge, MA, 1985.

10. Awerbuch, B. and Micali, S. "Dynamic Deadlock Resolution Protocols," *27th IEEE Symposium Foundations of Computer Science*, 1986, 196-207.

11. Barrington, D.A. "Width 3 Permutation Branching Programs," MIT/LCS/TM-293, MIT Laboratory for Computer Science, Cambridge, MA, December 1985.

12. Barrington, D.A. "Bounded Width Polynomial Size Branching Programs," *18th ACM Symposium Theory of Computing*, 1986, 1-5. Also invited for a special issue of *JCSS*, to appear 1987.

13. Barrington, D.A. "Bounded Width Branching Programs," Ph.D. dissertation, MIT Department of Mathematics, Cambridge, MA, 1986.

14. Ben-Or, M., Goldreich, O., Micali, S. and Rivest, R.L. "A Fair Protocol for Signing Contracts," *12th International Colloquium on Automata, Languages and Programming*, LNCS: 194, W. Brauer (ed.), Springer-Verlag, 1985, 43-52.

15. Bentley, J.L., Leighton, F.T., Lepley, M., Stanat, D. and Steele, M. "A Randomized Data Structure for Ordered Sets," *Advances in Computing Research*, S. Micali (ed.), 1986, to appear.

16. Berger, B. "New Upper Bounds for Two-Layer Channel Routing," S.M. thesis MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1986.

17. Berger, B., Brady, M., Brown, D. and Leighton, F.T. "An Almost Optimal Algorithm for Multilayer Channel Routing," *27th IEEE Symposium Foundations of Computer Science*, 1986.

18. Berman, F., Johnson, D.S., Leighton, F.T., Shor, P. and Snyder, L. "Generalized Planar Matching," *J. Algorithms*, (1986), to appear.

19. Bhatt, S., Chung, F.R.K., Leighton, F.T. and Rosenberg, A. "Optimal Simulations of Tree Machines," *27th IEEE Sympcsium Foundations of Computer Science*, 1986, 274-282.

20. Bloom, B. "Multi-writer Atomic Memory," unpublished.

21. Bloom, B. and Rivest, R. "Learning Automata," unpublished.

22. Boppana, R.B. "Amplification of Probabilistic Boolean Formulas," *26th IEEE Symposium Foundations of Computer Science*, 1985, 20-29.

23. Boppana, R.B. "Lower Bounds for Monotone Circuits and Formulas," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1986.

24. Boppana, R. B. and Hastad, J. "Approximation Properties of Constant Depth Circuits," to appear.

25. Boppana, R.B. and Hirschfeld, R. "Pseudorandom Generators and Complexity Classes," *Advances in Computing Research*, S. Micali, (ed.), 1986, to appear.

26. Boppana, R.B. and Lagarias, J.C. "One-way Functions and Circuit Complexity," *ACM Structure in Complexity Theory Conference*, LNCS: 223, Springer-Verlag, 1986, 51-65.

27. Breazu-Tannen, V. and Meyer, A.R. "Lambda Calculus with Constrained Types (Extended Abstract)," In *Logics of Programs*, LNCS: 193, R. Parikh (ed.), Springer-Verlag, 1985, 23-40.

28. Bruce, K., Meyer, A.R. and Mitchell, J.C. "The Semantics of Second-order Polymorphic Lambda Calculus," *Information and Control*, (1986), to appear. An earlier version authored by Bruce and Meyer appeared in Springer Lecture Notes in Computer Science, vol. 173.

29. Bui, T., Chaudhuri, S. Leighton, F.T. and Sipser, M. "Graph Bisection Algorithms with Good Average Case Behavior," *Combinatorica*, (1986), to appear.

30. Buss, J. "Relativized Alternation and Space-Bounded Computation," Ph.D. dissertation, MIT Department of Mathematics, Cambridge, MA, 1986.

31. Chor, B. "Two Issues in Public-Key Cryptography," Cambridge, MA, MIT Press, 1986. Also ACM Distinguished Computer Science Ph.D. dissertation.

32. Chor, B., Friedmann, J., Goldreich, O., Hastad, J., Rudich, S. and Smolensky, R. "The Bit Extraction Problem, or T-resilient Functions," *26th IEEE Symposium Foundations of Computer Science*, 1985, 396-407.

33. Chor, B., Goldwasser, S., Micali, S. and Awerbuch, B. "Verifiable Secret

Sharing and Simultaneous Broadcast," *26th IEEE Symposium Foundations of Computer Science*, 1985, 383-395.

34. Chor, B., Leiserson, C., Rivest, R. and Shearer, J. "An Application of Number Theory to the Organization of Raster-graphics Memory," *Journal of the ACM*, 33, (1986), 86-104.

35. Chor, B. and Dwork, C. "Randomized Algorithms for Distributed Agreement," submitted.

36. Chor, B. and Goldreich, O. "Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity," *26th IEEE Symposium Foundations of Computer Science*, 1985, 429-442.

37. Chor, B. and Goldreich, O. "An Improved Parallel Algorithm for Integer GCD," *Algorithmica*, (1986), submitted.

38. Chor, B. and Goldreich, O. "On the Power of Two-point Based Sampling," *Journal of Complexity*, (1986), submitted.

39. Chor, B., Goldreich, O. and Goldwasser, S. "The Bit Security of Modular Squaring Given Partial Factorization of the Modulos," *Advances in Cryptology: Proceedings of Crypto85*, H. C. Williams (ed.), Springer Verlag, 1986, 448-457.

40. Chor, B., Merritt, M. and Shmoys, D.B. "Simple Constant-time Consensus Protocols in Realistic Failure Models," *4th ACM Symposium Principles of Distributed Computing*, R.H. Strong (ed.), 1985, 152-162.

41. Chung, F.R.K., Leighton, F.T. and Rosenberg, A. "Embedding Graphs in Books: A Layout Problem With Applications to VLSI Design," *SIAM Journal of Algebraic and Discrete Methods*, (1986), to appear.

42. Coffman, E.G., Kadota, T., Leighton, F.T. and Shepp, L. "Stochastic Analysis of Storage Fragmentation," *International Sem. Teletraffic Analysis and Computer Performance Evaluation*, North Holland, 1986, to appear.

43. Coffman, E. and Leighton, F.T. "A Provably Efficient Algorithm for Dynamic Storage Allocation," *18th ACM Symposium Theory of Computing*, 1986, 77-90.

44. Corman, T.H. and Leiserson, C.E. "A Hyperconcentrator Switch for Routing Bit-serial Messages," *15th IEEE Conference on Parallel Processing*, Penn. State Univ., 1986, 721-736.

45. Elias, P. "Interval and Recency-rank Source Coding: Two On-line Adaptive Variable-length Schemes," *IEEE Transactions on Information Theory*, (1986).

46. Even, S., Goldreich, O. and Shamir, A. "On the Security of Ping-pong Protocols When Implemented Using RSA," *Advances in Cryptology: Proceedings of Crypto85*, H.C. Williams (ed.), Springer Verlag, 1986, 58-72.

47. Feldman, P., Friedman, J. and Pippinger, N. "Non-blocking Networks," *18th ACM Symposium Theory of Computing*, 1986, 247-253.

48. Feldman, P. and Micali, S. "Byzantine Agreement in Constant Expected Time (and Trusting No One)," *26th IEEE Symposium Foundations of Computer Science*, 1985, 267-276.

49. Gilmore, P.C., Lawler, E.L. and Shmoys, D.B. "Well-solvable Cases of the Traveling Salesman Problem," in *The Traveling Salesman Problem*, by E. L. Lawler and Lenstra, J., A.H. G. Rinnooy Kan and D.B. Shmoys (eds.), J. Wiley and Sons, 1985, 87-143.

50. Goldberg, A.V. "A New Max-flow Algorithm," MIT/LCS/TM-291, MIT Laboratory for Computer Science, Cambridge, MA, November 1985.

51. Goldberg, A.V. and Tarjan, R.E. "A New Approach to the Maximum Flow Problem," *18th ACM Symposium Theory of Computing*, 1986, 136-146.

52. Goldreich, O., Micali, S. and Wigderson, A. "Proofs That Yield Nothing But Their Validity and a Methodology For Cryptographic Protocol Design," *27th IEEE Symposium Foundations of Computer Science*, 1986, 174-187.

53. Goldreich, O. and Shrira, L. "The Effect of Link Failures on Computations in Asynchronous Rings," *5th ACM Symposium Principles of Distributed Computing*, J. Halpern (ed.), 1986, 174-185.

54. Goldreich, O., Goldwasser, S. and Micali, S. "The Cryptographic Applications of Random Functions (Extended Abstract)," *Advances in Cryptology: Proceedings Crypto 84*, LNCS:196, G.R. Blackley and D. Chaum (eds.), Springer-Verlag, 1985, 276-288.

55. Goldreich, O., Goldwasser, S. and Micali, S. "How to Construct Random Functions," *Journal of ACM*, 33, 4, (1986), 792-807.

56. Goldwasser, S. and Kilian, J. "Almost All Primes Can Be Quickly Certified," *18th ACM Symposium Theory of Computation*, 1986, 316-329.

57. Goldwasser, S., Micali, S. and Rackoff, C. "The Knowledge Complexity of Interactive Proof Systems," *Journal of ACM*, (1986), submitted.

58. Goldwasser, S., Micali, S. and Rivest, R. "A Signature Scheme Which is Secure Against Chosen Cyphertext Attack," *Journal of ACM*, (1986), submitted.

59. Goldwasser, S. and Sipser, M. "Interactive Proof Systems: Public vs. Private Coins," *18th ACM Symposium Theory of Computation*, 1986, 59-68.

60. Greenberg, R.I. and Leiserson, C.E. "Randomized Routing on Fat-trees," *26th IEEE Symposium Foundations of Computer Science*, 1985, 241-249.

61. Halpern, J. Louis, M., Meyer, A. and Weise, D. "On Time Versus Space III," *MST*, 19, (1986), 13-28.

62. Hastad, J. "On Using RSA With Low Exponent in a Public Key Network," *SICOMP*, (1986), to appear.

63. Hastad, J. "Almost Optimal Lower Bounds for Small Depth Circuits," *18th ACM Symposium Theory of Computing*, 1986, 6-20.

64. Hastad, J. "Computational Limitations for Small Depth Circuits," Ph.D. dissertation, MIT Department of Mathematics, Cambridge, MA, 1986.

65. Hastad, J., Just, B., Lagarias, J.C. and Schnorr, C.P. "On Finding Integer Relations Among Real Numbers," *SICOMP*, (1986), submitted.

66. Hastad, J. and Leighton, F.T. "Division in O(log N) Depth Using O(N 1+ε) Processors," under revision for publication (1985).

67. Heath, L.S. "Algorithms for Embedding Graphs in Books," Report TR-85-028, University of North Carolina, Chapel Hill, NC, August 1985.

68. Heath, L.S. "Embedding Outerplanar Graphs in Small Books," *SIAM Journal of Algebraic and Discrete Methods*, (1985), to appear.

69. Heath, L.S. "Embedding Trivalent Planar Graphs in Two Pages," *SICOMP*, (1985), submitted.

70. Heath, L.S. and Istrail, S. "Surface-embeddable Graphs can be Embedded in a Bounded Number of Pages," *27th IEEE Symposium Foundations of Computer Science*, 1986.

71. Hirschfeld, R. "Pseudorandom Generators and Complexity Classes," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1986.

72. Hochbaum, D.S., Nishizeki, T. and Shmoys, D.B. "A Better Than 'Best Possible' Algorithm to Edge Color Multigraphs," *Journal of Algorithms*, 7, (1986), 79-104.

73. Hochbaum, D.S. and Shmoys, D.B. "An O(V) Algorithm for the Planar 3-cut Problem," *SIAM Journal of Algebraic and Discrete Methods*, 6, (1985), 707-712.

74. Hochbaum, D.S. and Shmoys, D.B. "Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results," *26th IEEE Symposium Foundations of Computer Science*, 1985, 79-89.

75. Hochbaum, D.S. and Shmoys, D.B. "A Packing Problem You Can Almost Solve by Sitting on Your Suitcase," *SIAM Journal of of Algebraic and Discrete Methods*, 7, (1986), 247-257.

76. Hochbaum, D.S. and Shmoys, D.B. "A Unified Approach to Approximation Algorithms for Bottleneck Problems," *Journal of ACM*, 33, (1986), 533-550.

77. Kaklamanis, C. "A Special Case of First-order Strictness Analysis," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1986.

78. Kaliski, B.S. "Wyner's Analog Encryption Scheme: Results of a Simulation," *Advances in Cryptology: Proceedings Crypto 84*, LNCS:196, G.R. Blakley and D. Chaum (eds.), Springer-Verlag, 1985, 83-94.

79. Kaliski, B.S., Rivest, R.L. and Sherman, A.T. "Is DES a Pure Cipher? (Results of More Cycling Experiments on DES)." *Proceedings of Crypto 85*, Springer-Verlag, 1985, 212-222.

80. Kaliski, B.S., Rivest, R.L. and Sherman, A.T. "Is the Data Encryption Standard a Group?" *Advances in Cryptology: Eurocrypt 85*, F. Pichler (ed.), Springer-Verlag, 1986, 81-95.

81. Karloff, H. J. and Shmoys, D.B. "Efficient Parallel Algorithms for Edge Coloring Problems," *Journal of Algorithms*, 7, (1986), to appear.

82. Karp, R.M., Leighton, F.T., Rivest, R., Thompson, C., Vazirani, U. and Vazirani, V. "Global Wire Routing in Two-dimensional Arrays," *Algorithmica*, 2, 1, (1986), 113-129.

83. Klein, P. and Reif, J. "An Efficient Parallel Algorithm for Planarity," S.M thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1986.

84. Klein, P. and Reif, J. "Parallel Time O(log n) Acceptance of Deterministic CFL's," *SICOMP*, (1985), submitted. Also available as TR-05-84, Center for Research in Computing Technology, Harvard University, Cambridge, MA.

85. Lagarias, J.C. and Hastad, J. "Simultaneous Diophantine Approximation of Rationals by Rationals," *Journal of Number Theory*, (1986), to appear.

86. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D. (eds.) "The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization," J. Wiley and Sons, 1985.

87. Leighton, F.T. "An Introduction to the Theory of Networks, Parallel Computation and VLSI Design," in progress, 1986.

88. Leighton, F.T. and Leiserson, C.E. "Wafer-scale Integration of Systolic Arrays," *IEEE Transactions on Computers*, C-34, (1985), 448-461.

89. Leighton, F.T. and Leiserson, C.E. "Algorithms for Integrating Wafer-scale Systolic Arrays," *VLSI Circuit and Architecture Design*, E. Swartzlander (ed.), Marcel-Dekker, 1986, to appear.

90. Leighton, F.T. and Leiserson, C.E. "A Survey of Algorithms for Integrating Wafer-scale Systolic Arrays," *IFIP Workshop on Wafer-Scale Integration at Grenoble*, G. Saucier and J. Truihle (eds.), North Holland, 1986, 177-195.

91. Leighton, F.T. and Rivest, R. "Estimating a Probability Using Finite Memory," *IEEE Transactions on Information Theory*, (1986), to appear.

92. Leighton, F.T. and Rosenberg, A. "Three-dimensional Circuit Layouts," *SICOMP*, (1986), to appear.

93. Leighton, F.T. and Shor, P. "Tight Bounds for Minimax Grid Matching, With Applications to the Average Case Analysis of Algorithms," *18th ACM Symposium Theory of Computing*, 1986, 91-103.

94. Leiserson, C.E. and Maggs, B.M. "Communication-efficient Parallel Graph Algorithms," *1986 International Conference Parallel Processing*, 1986, 861-868.

95. Levin, L.A. "Average Case Complete Problems," *SICOMP*, 15, (1986), 285-6.

96. Levin, L.A. "One Way Functions and Pseudorandom Generators," *Combinatorica*, (1986), to appear.

97. Maggs, B. "Communication-Efficient Parallel Graph Algorithms." S.M thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1986.

98. Malitz, S.M. "Measures of Graphs on the Reals," in preparation.

99. Meyer, A.R. Floyd-Hoare Logic Determines Semantics," *IEEE Symposium Logic in Computer Science*, 1986, 44-48.

100. Meyer, A.R. "Thirteen Puzzles in Programming Logic," *Proceedings DDC Workshop on Formal Software Development: Combining Specification*, LNCS, D. Bjorner, (ed.), Springer-Verlag, 1986.

101. Meyer, A.R. and Reinhold, M.B. "'Type' is Not A Type: Preliminary Report," *13th ACM Symposium Principles of Programming Languages*, 1986, 287-295.

102. Meyer, A.R. and Wand, M. "Continuation Semantics in Typed Lambda-calculi (Summary)," *Logics of Programs*, LNCS:193, R. Parikh (ed.), Springer-Verlag, 1985, 219-224.

103. Micali, S. "Knowledge and Efficient Computation," *1st ACM Symposium Theoretical Aspects of Reasoning About Knowledge*, J. Halpern (ed.), 1986, 353-362.

104. Micali, S., Fischer, M., Rackoff, C. and Wittenberg, D. "An Oblivious Transfer Protocol," in preparation.

105. Micali, S., Galil, Z. and Gabow, H. "An O(E V log V) Algorithm for Finding a Maximal Weighted Matching in General Graphs," *SICOMP*, 15, (1986), 120-130.

106. Micali, S., Rackoff, C. and Sloan, B. "The Notion of Security for Probabilistic Cryptosystems," *SICOMP*, (1986), submitted.

107. Mitchell, J.C. and Meyer, A.R. "Second-order Logical Relations (Extended Abstract)," *Logics of Programs*, LNCS:193, R. Parikh (ed.), Springer-Verlag, 1985, 225-236.

108. Parikh, R., Chandra, A., Halpern, J. and Meyer, A.R. "Equations Between Regular Terms and an Application to Process Logic," *SIAM Journal of Computing*, 14, 4, (1985), 935-94.

109. Phillips, C.A. "Space-Efficient Algorithms for Computational Geometry," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1985.

110. Rivest, R. L. "Network Control by Bayesian Broadcast," MIT/LCS/TM-287, MIT Laboratory for Computer Science, Cambridge, MA, September 1985.

111. Sherman, A.T. "Cryptology and VLSI (A Two-part Dissertation): I. Detecting and Exploiting Algebraic Weaknesses in Cryptosystems; II. Algorithms for Placing Modules on a VLSI Chip," MIT/LCS/TR-381, MIT Laboratory for Computer Science, Cambridge, MA, 1986.

112. Sipser, M. "Expanders, Randomness, and Time versus Space," *Conference in Structure in Complexity Theory*, 1986, 325-329.

113. Trakhtenbrot, B.A. "Selected Developments in Soviet Mathematical Cybernetics," Delphic Associates, Monograph Series on Soviet Union, Falls Church, VA, 1986.

114. Trakhtenbrot, B.A. "Logical Relations in Program Semantics," *Conference Mathematical Logic and its Applications*, Plenum Press, 1986.

115. Vitanyi, P. and Awerbuch, B. "Atomic Shared Register Access by Asynchronous Hardware," *27th IEEE Symposium Foundations of Computer Science*, 1986, 233-243.

Talks

1. Barrington, D.A. "Bounded Width Polynomial Size Branching Programs Recognize Exactly Those Languages in NC^1," 18th ACM Symposium Theory of Computing; MIT Laboratory for Computer Science; IBM Research Center, San Jose; Mathematical Sciences Research Institute, Berkeley; U. Chicago; Wesleyan U.; U. Mass., Amherst; Amherst College; U. Toronto; U. Montreal, 1986.
2. Berger, B. and Leighton, T. "New Bounds and Algorithms for Channel Routing," Fall 1985 MIT VLSI Research Review, 1985.
3. Bloom, B. "Lectures on Denotational Semantics of Countable Nondeterminism," a series of lectures, MIT LCS Theory of Computation Concurrency Seminar, Cambridge, MA, 1986.
4. Bloom, B. and Rivest, R.L. "Abstract AI," Workshop in Machine Learning, Skytop, PN, 1985.
5. Boppana, R.B. "Lower Bounds for Monotone Circuits," MIT Laboratory for Computer Science, Cambridge, MA, and Yale University, 1985.
6. Boppana, R.B. "Amplification of Probabilistic Boolean Formulas," 26th FOCS; Bell Communications Research, 1985.
7. Breazu-Tannen, V. "Conservative Extension Situations in Typed Lambda Calculi," Bell Labs; IBM Yorktown Heights; Brown U.; Cornell University, 1986.
8. Chor, B., Goldwasser, S., Micali, S. and Awerbuch, B. "Verifiable Secret

Sharing and Simultaneous Broadcast," Workshop on Security, MIT Endicott House, Dedham, MA, 1985.

9. Chor, B. and Goldreich, O. "Unbiased Bits From Sources of Weak Randomness and Probabilistic Communication Complexity," Harvard U.; U. Toronto; Tel-Aviv U.; IBM, Yorktown; IEEE Symposium Foundations of Computer Science, 1985-86.

10. Chor, B. and Goldreich, O. "Efficient Pseudo-random Bits Generators Based on Factoring/RSA," Marseilles, Workshop on Algorithms, Randomness and Complexity, 1986.

11. Chor, B. and Goldreich, O. "Unbiased Bits From Sources of Weak Randomness and Probabilistic Communication Complexity," MIT Laboratory for Computer Science (1985); MSRI, UC-Berkeley; IBM San-Jose Research Center; Marseilles, workshop on Algorithms, Randomness and Complexity, 1985.

12. Cormen, T.H. and Leiserson, C.E. "A Hyperconcentrator Switch for Routing Bit-serial Messages," Darpa VLSI Contractor's Review, University of Utah, 1985.

13. Goldreich, O., Micali, S. and Wigderson, A. "Proofs That Yield Nothing But Their Validity, or All Languages in NP Have Zero-knowledge Proofs," MIT Laboratory for Computer Science and Yale University, 1986.

14. Goldreich, O., Micali, S. and Wigderson, A. "Methodological Theorems for Cryptographic Protocol Design," University of Toronto, 1986.

15. Goldwasser, S. "Encryption and Signatures in Public Key Cryptography," Series of talks given at a course on cryptography held in Amsterdam, 1985.

16. Goldwasser, S. and Sipser, M. "Interactive Proof Systems: Public vs. Private Coins," Yale; Brown; ACM STOC, 1986.

17. Goldwasser, S. and Kilian, J. "A Provably Correct and Probably Fast Primality Test," NYU; Eighth Columbia Theory Day, 1985.

18. Goldwasser, S. and Kilian, J. "Almost All Primes Can be Quickly Certified," U. of Toronto; Marseilles Workshop on Algorithmic Randomness and Complexity, 1985.

19. Greenberg, R.I. "Randomized Routing on Fat-trees," MIT; 1985 IEEE Symposium Foundations of Computer Science, 1985.

20. Hastad, J. "Almost Optimal Lower Bounds for Small Depth Circuits," MIT; IBM, Yorktown Heights; CIRM, Marseilles; MSRI, Berkeley; IBM; San Jose; UCLA; U. Toronto; Bell Communication Research, 1985.

21. Hastad, J. "The Bit-extraction Problem or t-resilient Functions," 26th IEEE Symposium Foundations of Computer Science, 1985.

22. Hastad, J. "On Finding Integer Relations Among Real Numbers," Conference in Computational Number theory at Arcata, 1985.

23. Hastad, J. "On Using RSA With Low Exponent in a Public Key Network," Crypto 1985, Santa Barbara, CA, 1985.

24. Heath, L. "Algorithms for Embedding Graphs in Books," Minisymposium on Book Embeddings at the 3rd SIAM Conference on Discrete Mathematics, 1986.

25. Kaliski, B., Rivest, R.L. and Sherman, A. "Is DES a Pure Cipher? (Results of More Cycling Experiments on DES)," CRYPTO 1985, Santa Barbara, CA, 1985.

26. Kaliski, B.S. "A Pseudo-random Bit Generator Based on Elliptic Logarithms," MIT Laboratory for Computer Science, Cambridge, MA, 1985.

27. Kaliski, B.S. "Lenstra's Factoring Algorithm Using Elliptic Curves," Tufts University, Medford, MA, 1985.

28. Levin, L.A."Homogeneous Measures and Polynomial Time Invariants," French Mathematics Society, Workshop on Algorithms, Randomness and Complexity, Marseilles, 1986.

29. Malitz, S.M. "Measures of Graphs on the Reals," MIT; 304th AMS Conference, Johns Hopkins University, 1986.

30. Meyer, A.R. "The Complexity of Flow-analysis: Application of a Fundamental Theorem of Denotational Semantics," Department of Computer Science, University of Pisa, Italy, 1985.

31. Meyer, A.R. "Floyd-Hoare Logic Determines Semantics," Presentation of paper at IEEE Symposium on Logic in Computer Science, Cambridge, MA, June 1986.

32. Meyer, A.R. "Axiomatic Semantics of Programs by Floyd-Hoare Logic," Yale University, 1986.

33. Meyer, A.R. "Types in Programming, An Overview," Invited Tutorial Lecture, 13th ACM Symposium Principles of Programming Languages, January 1986.

34. Meyer, A.R. "Logical Puzzles in Programming," Wesleyan University, CN, 1986.

35. Rivest, R.L. "The RSA Public-key Cryptosystem," Cryptography Workshop in Amsterdam (organized by David Chaum), 1985.

36. Sherman, A.T. "Algorithms for Placing Modules on a Custom VLSI Chip," MIT VLSI Research Review, Cambridge, MA, 1986.

37. Shmoys, D.B. "A Packing Problem You Can Almost Solve by Sitting on Your Suitcase," ORSA/TIMS meeting, Los Angeles, CA, 1986.

38. Shmoys, D.B. "Efficient Parallel Algorithms for Edge Coloring Problems," Mathematical Programming Society Symposium, Boston; Department of Mathematics, MIT; 1985.

39. Shmoys, D.B. "Using Dual Approximation Algorithms for Scheduling Problems," IEEE Symposium Foundations of Computer Science; Princeton, 1985.

40. Shmoys, D.B. "Simple Constant-time Consensus Protocols," ACM Conference Principles of Distributed Computing; Univ. Washington, Seattle; University of Chicago,1986.

41. Trakhtenbrot, B.A. "A Characterization Theorem for Program Schemes," Columbia University, 1986.

42. Trakhtenbrot, B.A. "Survey on Type Theory," University of Tennessee, 1986.

43. Trakhtenbrot, B.A. "On Semantics of Storage Allocation," University of Tennessee, 1986.

44. Trakhtenbrot, B.A. "Logical Relations Over FLAT and Sequentiality of Functionals," Series of lectures at MIT Laboratory for Computer Science Theory of Computation Seminar on Types in Programming, 1986.

THEORY OF DISTRIBUTED SYSTEMS

Academic Staff

N. A. Lynch, Group Leader

Graduate Students

B. Bloom
B. Coan
C. Clifton
K. Goldman
J. Lundelius
M. Tuttle

Support Staff

E. Pothier

Visitors

Y. Moses
P. Vitanyi

1. INDIVIDUAL PROGRESS REPORTS

Chris Clifton

C. Clifton's Master's thesis on "Dynamic Load Balancing", supervised by N. Lynch and F. Cristian (IBM Almaden Research Lab) is nearing completion. This thesis presents a new model for load in a computer system which separates different types of processing resources. This model is used to define the problem of load balancing in distributed systems, and a simple algorithm for load balancing is given.

While at IBM, Chris developed a system which performed load balancing on-line in a cluster of IBM mainframe processors. This system looks at CPU, Memory, and I/O as separate resources, and places tasks at different processors so as to avoid overloading any of these three resources.

B. Coan

During the past year, I have worked on the following:

(1) "A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols" [3]

Many fault-tolerant distributed protocols are known. Some of these require a large (exponential) amount of communication. We present a general simulation of any synchronous fault-tolerant consensus protocol by a communication-efficient protocol. An important corollary of the simulation technique is a new communication-efficient Byzantine agreement protocol that uses about half the number of rounds required by the best previously-known communication-efficient Byzantine agreement protocol. Our new protocol approaches the known lower bound for rounds to within a small factor arbitrarily close to 1. The only known protocols which achieve the lower bound for rounds use an exponential amount of communication.

(2) "Simultaneity is Harder than Agreement" (with C. Dwork) [5]

For many distributed problems, such as the distributed firing squad problem or the Byzantine agreement problem, it is known that $t+1$ is a lower bound on the number of rounds of message exchange required by any protocol that is resilient to t faults. Because this is a worst-case lower bound, there may well be a protocol that does better much of the time. Such a protocol is said to be *fast.* In this paper it is shown that, assuming fail-stop faults, there are no fast protocols (either randomized or deterministic) for the distributed firing squad problem or for the simultaneous weak Byzantine agreement problem. These results contrast with known fast protocols for the eventual Byzantine agreement problem. The observed pattern is that problems which require simultaneous termination of correct processors have no fast protocols, while problems which admit staggered termination of correct processors do have fast protocols. The

impossibility results of this paper are for a benign failure model and therefore apply *a fortiori* to more malicious failure models.

(3) "Transaction Commit in a Realistic Fault Model" (with J. Lundelius) [4]

We study the transaction commit problem under realistic timing assumptions. We identify an almost asynchronous model, which we claim is more realistic than some (synchronous) models that have been studied previously. In this model we give a randomized transaction commit protocol based on Ben-Or's randomized asynchronous Byzantine agreement protocol. The expected number of asynchronous rounds until our protocol terminates is a small constant, and the number of failstop faults tolerated is optimal. It is known that no deterministic protocol is possible in this model. We motivate our definition of asynchronous rounds by showing that no protocol in this model can terminate in a bounded expected number of clock ticks, even if processors are synchronous. Defining asynchronous rounds allows us to make the performance guarantee that after a sufficient number of useful messages have been delivered our protocol will terminate.

(4) "Limitations on Database Availability When Networks Partition" (with B. Oki and E. Kolodner) [6]

In designing fault-tolerant distributed database systems, a frequent goal is making the system highly available despite component failure. We examine software approaches to achieving high availability in the presence of partitions. In particular, we consider various replicated-data management protocols that maintain database consistency and attempt to increase database availability when networks partition. We conclude that no protocol does better than a bound we have determined. Our conclusions hold under the assumption that the pattern of data accesses by transactions obeys a uniformity assumption. There may be some particular distribution for which specialized protocols can increase availability.

K. Goldman

I have been examining various algorithms for managing replicated data. The hope is that through comparing these methods we can determine the basic mechanisms needed to model replicated data using the I/O Automata Model given by [18] and [17]. After doing so, it should be possible to prove properties of these methods and possibly derive better methods for handling replicated data.

J. Lundelius

This year my research concerned the impact of synchrony in distributed systems. I worked on three different projects.

B. Coan and I [4] investigated the transaction commit problem under realistic timing

assumptions. We described an almost asynchronous model that is more realistic than some synchronous models that have been studied previously. We designed a randomized transaction commit protocol for this model, based on Ben-Or's randomized asynchronous Byzantine agreement protocol [2]. We proved that the expected number of asynchronous rounds until our protocol terminates is a small constant, and that the number of failstop faults tolerated is optimal. It is known that no deterministic protocol is possible in this model. We also devised a new definition of asynchronous round that allows us to make the performance guarantee that after sufficiently many useful messages have been delivered, our protocol terminates. Additional justification for our definition is provided by our proof that no protocol in this model can terminate in a bounded expected number of clock ticks, even if processors are synchronous.

Another research effort was showing how a distributed system with synchronous processors and asynchronous message delays can be simulated by a system with both asynchronous processors and asynchronous message delays in the presence of various types of processor faults [15]. Consequently, the result of [10], that no consensus protocol for asynchronous processors and communication can tolerate one failstop fault, implies a result of [8], that no consensus protocol for synchronous processors and asynchronous communication can tolerate one failstop fault.

Finally, I also studied the problem of achieving consensus in various shared memory models of computation [16]. The impetus to work on this problem was provided by the manuscript [13]. I showed that there can be no fault-tolerant consensus protocol, for a very weak notion of consensus, if memory cells cannot be read and written in one atomic step. When atomic read/write is available, there is a maximally fault-tolerant weak consensus protocol, and a strong consensus protocol that tolerates one fault -- both these algorithms are described. The final result in this paper showed that even if atomic read/write is available, there is no strong consensus protocol that can tolerate three faults. (The paper [14], which is a revised version of [13], extends this result to show no protocol can tolerate two faults.)

N. A. Lynch

During this year, a theme has emerged in my work - that of trying to understand ways of decomposing and combining distributed algorithms. A problem with much of the work in this area so far has been that results tend to be isolated; it would be a great contribution to provide useful methods for building distributed algorithms out of subalgorithms. The methods I am interested in should be both rigorous and intuitive.

Of course, this is a very general goal. The actual day-to-day work involves intensive study of three specific application areas: distributed consensus, distributed concurrency control and distributed graph algorithms. In each area, I have been trying to organize

and generalize the important ideas using various decomposition techniques, as well as trying to discover new algorithms and lower bound results.

In the area of distributed consensus, I had a paper [22] at last summer's Symposium on Principles of Distributed Computing, which unified, strengthened, and simplified a large number of results in the recent literature about the impossibility of solving certain consensus problems in particular graph networks. Also, I have been supervising work by several Ph.D. students, on algorithms and lower bounds for consensus problems. B. Coan has obtained a very interesting set of algorithmic results; he uses several kinds of decomposition, including translating results from one model of distributed computation to another, and allowing an algorithm to call distributed subroutines to solve subproblems. J. Lundelius also uses translation ideas, in her case to obtain a simple proof of a difficult impossibility result in the literature, by reducing another problem, known to be unsolvable, to it. A. Fekete, a graduate student at Harvard, has also been working with me on fast algorithms for approximate agreement.

Most of my effort this year has been devoted to concurrency control issues; nested transactions seem to be indispensable as a basis for distributed programming languages, and there has so far been no usable general theory developed. A major project in progress, with M. Merritt, involves establishing a simple, rigorous and intuitive model for concurrent transaction processing. We have "nearly" completed a grand work which establishes the general model and proves correctness of some interesting practical algorithms [18]. Since other people have also gotten quite interested in applying this model to this work, we have also begun work with B. Weihl, M. Herlihy, and students K. Goldman and S. Perl, on describing orphan elimination algorithms, replicated data, and abstract atomic types. We should have more to report on this next year.

A different style of work on concurrent processing has arisen from consulting work at Computer Corporation of America, on the SHARD system. The SHARD system manages replicated data in a way that yields high availability, in spite of unreliable communication. The usual correctness condition, serializability, is not guaranteed, however. Then the question arises as to exactly what this system does guarantee. We give a partial answer to this question; for a particular kind of application (resource-allocation), we prove interesting statements about the cost bound and fairness properties which SHARD guarantees. It seems that this type of system is likely to become very important in real distributed database applications, and this work seems very promising as a start to understanding what such a system can do.

In the area of distributed graph algorithms, I have been working with Leslie Lamport recently, on a multi-level proof of correctness for a very difficult and well-known

distributed algorithm, Gallager, Humblet and Spira's algorithm for finding a minimum spanning tree. The main effort here is in disentangling the many ideas used in the algorithm, so that they can be understood one at a time.

Closely related to all of the work described above has been work with student M. Tuttle on models for distributed algorithms. We believe we have settled on a general model which seems appropriate to all types of concurrent algorithms. In [17], we are using it to provide a multi-level correctness proof for a network resource allocation algorithm, but we are also using it elsewhere to describe concurrency control algorithms, shared memory algorithms, dataflow algorithms, etc.

Some papers on older work ([1],[7]), were also submitted (or revised and resubmitted) during this year.

Y. Moses

Y. Moses joined our group on September 30th as a post-doctoral fellow for a year. His main research effort since then has been in understanding how reasoning about knowledge can be used as an essential tool in analyzing and understanding complex problems in distributed systems. He has since written two papers on the relationship between simultaneous actions and various states of knowledge in systems of unreliable processors (the much studied but still mysterious subject of Byzantine agreement and its variants).

The first paper [9] was written with C. Dwork, formerly from our group but now from IBM Almaden Research center. A short abstract of the paper follows:

By analyzing the states of knowledge that the processors attain in an unreliable system of a simple type, we capture some of the basic underlying structure of such systems. In particular, we study what facts become *common knowledge* at various points in the execution of protocols in an unreliable system. This characterizes the simultaneous actions that can be carried out in such systems. For example, we obtain a complete characterization of the number of rounds required to reach *simultaneous byzantine agreement*, given the pattern in which failures occur. From this we derive a new protocol for this problem that is optimal *in all runs*, rather than just always matching the worst-case lower bound. In some cases this protocol attains Simultaneous Byzantine Agreement in as few as 2 rounds. We also present a non-trivial simultaneous agreement problem called *bivalent agreement* for which there is a protocol that always halts in two rounds. Our analysis applies to simultaneous actions in general, and not just to Byzantine agreement. The lower bound proofs presented here generalize and simplify the previously known proofs.

The second paper [21], with M. Tuttle of our group, extends the analysis of the first paper to the omissions model, again deriving efficient optimal protocols for

simultaneous actions in cases that is possible, and carefully delineating the boundary at which optimal protocols become computationally infeasible. A short abstract follows:

We study the problem of designing protocols that perform simultaneous actions in systems of unreliable processors in an optimal number of rounds in all runs and not merely matching lower bounds on performance in worst case runs. Simultaneous actions are closely related to common knowledge. We develop clean and efficient methods for determining what facts are common knowledge in several variants of the omissions failure model of practical interest. These methods yield computationally-efficient protocols for performing simultaneous actions in an optimal number of rounds in all runs. Conversely, we are able to show that, in the slightly more malevolent generalized omissions model, determining when common knowledge of many relevant facts is attained is inherently intractable (assuming P is different from NP). Hence, no optimal protocol for performing simultaneous actions in the generalized omissions model can be computationally efficient. This shows that, in models at least as malevolent as the generalized omissions model, common knowledge no longer directly corresponds to the simultaneous actions that resource-bounded processors are able to perform. In light of this, we suggest a notion of polynomial time common knowledge which does correspond to when simultaneous actions can be performed in systems of unreliable, resource-bounded processors.

M. Tuttle

My work this year has been concentrated in two areas. First, I am finishing my Master's thesis entitled "Correctness Proofs for Distributed Algorithms" in which the problem of developing natural proofs of the correctness and complexity of distributed network algorithms is addressed, where a "natural proof" is one that retains the flavor of an explanation one might give to a colleague at the blackboard of how an algorithm works. This work has been done in collaboration with my thesis advisor, N. Lynch. In this thesis we formalize a model of computation called the input-output automaton. One original aspect of this model is the division of process actions into input actions and output actions, and with this partitioning of actions we are able to give a useful notion of fairness. Using these automata we develop a style of verification that encourages proofs involving several conceptual levels of abstraction and encourages a modular decomposition of proofs.

Second, I am in the process of completing work with Y. Moses entitled "Common Knowledge and Simultaneous Actions in Systems of Unreliable Processors." In this paper we use the language of knowledge to study when processors in a distributed system of unreliable processors can perform actions simultaneously. We are able to completely characterize when certain types of simultaneous actions can be performed in a distributed system in the presence of omission failures, and this characterization

leads to computationally-efficient protocols for performing simultaneous actions in an optimal number of rounds in all runs and not just matching lower-bound performance in worst case runs. Conversely, we are able to show that, in the presence of the more malevolent generalized omission failures, determining when common knowledge of simple facts is attained is inherently intractable, and hence that optimal protocols for simultaneous actions are computationally infeasible. This work has the effect of pointing out subtle differences between the different models of failure in distributed systems. In addition, this work suggests a notion of polynomial time common knowledge which corresponds to when simultaneous actions can be performed in systems of unreliable, resource-bounded processors.

Publications

1. Burns, J. E. and Lynch, N. A. "The Byzantine Firing Squad Problem," 1986, submitted.

2. Ben-Or, M. "Fast Asynchronous Byzantine Agreement," *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing,* Minaki, Ontario, Canada, August 1985, 149-151.

3. Coan, B. "A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols," *Proceedings of the Fifth ACM SIGACT-SIGOPS Principles of Distributed Computing,* August 1986, 63-72.

4. Coan, B. and Lundelius, J. "Transaction Commit in a Realistic Fault Model," *Proceedings of the Fifth ACM SIGACT-SIGOPS Principles of Distributed Computing,* August 1986, 40-51.

5. Coan, B. and Dwork, C. "Simultaneity is Harder than Agreement," *Proceedings of the Fifth IEEE Symposium on Reliability in Distributed Software and Database Systems,* January 1986.

6. Coan, B., Oki, B. and Kolodner, E. "Limitations on Database Availability When Networks Partition," *Proceedings of the Fifth ACM SIGACT-SIGOPS Principles of Distributed Computing,* August 1986, 187-194.

7. Dolev, D., Lynch, N., Pinter, S., Stark, E. and Weihl, W. "Reaching Approximate Agreement in the Presence of Faults," *Proceedings of 3rd Symposium on Reliability in Distributed Software and Database Systems,* 1983, 145-154.

8. Dolev, D., Dwork, C. and Stockmeyer, L., "On the Minimal Synchronism Needed for Distributed Consensus," *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science,* 393-402, 1983.

9. Dwork, C., and Moses, Y. "Knowledge and Common Knowledge in a Byzantine Environment I: The Case of Crash Failures," *Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge,* J.Y. Halpern (ed.), Monterey, CA, March 1986, 149-169. To appear as an MIT Technical report.

10. Fischer, M., Lynch, N. and Paterson, M. "Impossibility of distributed Consensus with One Faulty Process," *JACM,* 32, 2, (1985), 374-382.

11. Frederickson, G. and Lynch, N. "Electing a Leader in a Synchronous Ring," MIT/LCS/TM-277, MIT Laboratory for Computer Science, Cambridge, MA, July 1985.

12. Lamport, L. and Lynch, N. "A Hierarchical Proof of a Distributed Algorithm," in progress.

13. Loui, M. and Amara, H. A. "Shared Memory Requirements for Agreement Among Unreliable Asynchronous Processors," manuscript, January 1985.

14. Loui, M. and Amara, H. A. "Shared Memory Requirements for Agreement Among Unreliable Asynchronous Processors," manuscript, April 1986.

15. Lundelius, J. "Simulating Synchronous Processors," manuscript, May 1986.

16. Lundelius, J. "Fault-Tolerant Consensus in a Shared Memory Model," manuscript, April 1986.

17. Lynch, N. and Tuttle, M. "Correctness Proofs for Distributed Algorithms," work in progress.

18. Lynch, N. and Merritt, M. "The Theory of Nested Transactions: Concurrency Control and Resiliency," 1986, submitted to a conference.

19. Lynch, N., Blaustein, B. and Siegel, M. "Correctness Conditions for Highly Available Replicated Databases," *Proceedings of the Fifth ACM SIGACT-SIGOPS Principles of Distributed Computing*, August 1986, 11-28.

20. Moses, Y., Dolev, D. and Halpern, J. Y. "Cheating Husbands and Other Stories: A Case Study of Knowledge, Action, and Communication," *Journal of Distributed Computing*, 1, 3, 167-176.

21. Moses, Y. and Tuttle, M. "Programming Simultaneous Actions Using Common Knowledge," MIT/LCS/TR-369, MIT Laboratory for Computer Science, Cambridge, MA, 1986.

22. Fischer, M. J., Lynch, N. A. and Merritt, M. "Easy Impossibility Proofs for Distributed Consensus Problems," *Proceedings of the Fourth Symposium on Principles of Distributed Computing*, 1985, 59-70 Minaki, Ontario, Canada August "Invited to appear in the first issue of a new journal, Distributed Computing, 26-39."

Theses in Progress

1. Clifton, C. "Dynamic Load Balancing," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, MA, to appear.

2. Coan, B. A. "Fundamental Problems in Fault-Tolerant Distributed Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, completion expected May 1986.

3. Tuttle, M. R. "Correctness Proofs for Distributed Algorithms," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, MA, to appear.

Talks

1. Coan, B. "Simultaneity is Harder than Agreement," Fifth Annual IEEE Symposium on Reliability in Distributed Software and Database Systems, Los Angeles, CA, January 1986.

2. Coan, B. "A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols,"

Bell Communications Research, Morristown, NJ, April 1986
AT&T Bell Labs, Murray Hill, NJ, April 1986.

3. Lynch, N. "Easy Impossibility Proofs for Distributed Consensus Problems,"

   Purdue University, November 1985
   Indiana University, November 1985
   University of Maryland, October 1985
   Bell Core, April 1986.

4. Lynch, N. "Clock Synchronization," Workshop on Fault-Tolerant Distributed Computing, Asilomar, CA, March 1986.

5. Moses, Y. "Knowledge and Common Knowledge in a Distributed Environment," GTE Labs.

6. Moses, Y. "Knowledge and Common Knowledge in a Byzantine Environment I: The Case of Crash Failures,"

   Workshop on Fault-Tolerant Distributed Computing, Asilomar, CA, March 1986
   MIT
   Cornell
   AT&T Bell Labs
   Harvard
   University of Toronto.

PUBLICATIONS

Technical Memoranda

TM-10[5]	Jackson, J.N. Interactive Design Coordination for the Building Industry, June 1970, AD 708-400
TM-11	Ward, P.W. Description and Flow Chart of the PDP-7/9 Communications Package, July 1970; AD 711-379
TM-12	Graham, R.M. File Management and Related Topics June 12, 1970, September 1970; AD 712-068
TM-13	Graham, R.M. Use of High Level Languages for Systems Programming, September 1970; AD 711-965
TM-14	Vogt, C.M. Suspension of Processes in a Multi-processing Computer System, September 1970; AD 713-989
TM-15	Zilles, S.N. An Expansion of the Data Structuring Capabilities of PAL, October 1970; AD 720-761
TM-16	Bruere-Dawson, G. Pseudo-Random Sequences, October 1970; AD 713-852
TM-17	Goodman, L.I. Complexity Measures for Programming Languages, September 1971, AD 729-011
TM-18	Reprinted as TR-85
TM-19	Fenichel, R.R. A New List-Tracing Algorithm, October 1970; AD 714-522
TM-20	Jones, T.L. A Computer Model of Simple Forms of Learning, January 1971; AD 720-337
TM-21	Goldstein, R.C. The Substantive Use of Computers For Intellectual Activities, April 1971, AD 721-618
TM-22	Wells, D.M. Transmission Of Information Between A Man-Machine Decision System And Its Environment, April 1971, AD 722-837

[5]TMs 1-9 were never issued.

TM-23	Strnad, A.J. The Relational Approach to the Management of Data Bases, April 1971; AD 721-619
TM-24	Goldstein, R.C. and Strnad, A.J. The MacAIMS Data Management System, April 1971, AD 721-620
TM-25	Goldstein, R.C. Helping People Think, April 1971, AD 721-998
TM-26	Iazeolla, G.G. Modeling and Decomposition of Information Systems for Performance Evaluation, June 1971, AD 733-965
TM-27	Bagchi, A. Economy of Descriptions and Minimal Indices, January 1972, AD 736-960
TM-28	Wong, R. Construction Heuristics for Geometry and a Vector Algebra Representation of Geometry, June 1972, AD 743-487
TM-29	Hossley, R. and Rackoff, C. The Emptiness Problem for Automata on Infinite Trees, Spring 1972; AD 747-250
TM-30	McCray, W.A. SIM360: A S/360 Simulator, October 1972; AD 749-365
TM-31	Bonneau, R.J. A Class of Finite Computation Structures Supporting the Fast Fourier Transform, March 1973; AD 757-787
TM-32	Moll, R. An Operator Embedding Theorem for Complexity Classes of Recursive Functions, May 1973; AD 759-999
TM-33	Ferrante, J. and Rackoff, C. A Decision Procedure for the First Order Theory of Real Addition with Order, May 1973; AD 760-000
TM-34	Bonneau, R.J. Polynomial Exponentiation: The Fast Fourier Transform Revisited, June 1973, PB 221-742
TM-35	Bonneau, R.J. An Interactive Implementation of the Todd-Coxeter Algorithm, December 1973; AD 770-565
TM-36	Geiger, S.P. A User's Guide to the Macro Control Language, December 1973; AD 771-435
TM-37	Schonhage, A. Real-Time Simulation of Multidimensional Turing Machines by Storage Modification Machines, December 1973; PB 226-103/AS

TM-38	Meyer, A.R. Weak Monadic Second Order Theory of Successor Is Not Elementary-Recursive, December 1973; PB 226-514/AS
TM-39	Meyer, A.R. Discrete Computation: Theory and Open Problems, January 1974; PB 226-836/AS
TM-40	Paterson, M.S., Fischer, M.J. and Meyer, A.R. An Improved Overlap Argument for On-Line Multiplication, January 1974, AD 773-137
TM-41	Fischer, M.J. and Paterson, M.S. String-Matching and Other Products, January 1974; AD 773-138
TM-42	Rackoff, C. On the Complexity of the Theories of Weak Direct Products, January 1974, PB 228-459/AS
TM-43	Fischer, M.J. and Rabin, M.O. Super-Exponential Complexity of Presburger Arithmetic, February 1974, AD 775-004
TM-44	Pless, V. Symmetry Codes and their Invariant Subcodes, May 1974; AD 780-243
TM-45	Fischer, M.J. and Stockmeyer, L.J. Fast On-Line Integer Multiplication, May 1974; AD 779-889
TM-46	Kedem, Z.M. Combining Dimensionality and Rate of Growth Arguments for Establishing Lower Bounds on the Number of Multiplications, June 1974, PB 232-969/AS
TM-47	Pless, V. Mathematical Foundations of Flip-Flops, June 1974, AD 780-901
TM-48	Kedem, Z.M. The Reduction Method for Establishing Lower Bounds on the Number of Additions, June 1974, PB 233-538/AS
TM-49	Pless, V. Complete Classification of (24,12) and (22,11) Self-Dual Codes, June 1974, AD 781-335
TM-50	Benedict, G.G. An Enciphering Module for Multics, S.B. Thesis, EE Department, July 1974, AD 782-658
TM-51	Aiello, J.M. An Investigation of Current Language Support for the Data Requirements of Structured Programming, S.M. & E.E. Thesis, EE Department, September 1974; PB 236-815/AS
TM-52	Lind, J.C. Computing in Logarithmic Space, September 1974, PB 236-167/AS

TM-53 Bengelloun, S.A.
MDC-Programmer: A Muddle-to-Datalanguage Translator for Information Retrieval, S.B. Thesis, EE Department, October 1974, AD 786-754

TM-54 Meyer, A.R.
The Inherent Computation Complexity of Theories of Ordered Sets: A Brief Survey, October 1974, PB 237-200/AS

TM-55 Hsieh, W.N., Harper, L.H. and Savage, J.E.
A Class of Boolean Functions with Linear Combinatorial Complexity, October 1974, PB 237-206/AS

TM-56 Gorry, G.A.
Research on Expert Systems, December 1974

TM-57 Levin, M.
On Bateson's Logical Levels of Learning, February 1975

TM-58 Qualitz, J.E.
Decidability of Equivalence for a Class of Data Flow Schemas, March 1975, PB 237-033/AS

TM-59 Hack, M.
Decision Problems for Petri Nets and Vector Addition Systems, March 1975; PB 231-916/AS

TM-60 Weiss, R.B.
CAMAC: Group Manipulation System, March 1975, PB 240-495/AS

TM-61 Dennis, J.B.
First Version of a Data Flow Procedure Language, May 1975

TM-62 Patil, S.S.
An Asynchronous Logic Array, May 1975

TM-63 Pless, V.
Encryption Schemes for Computer Confidentiality, May 1975, AD A010-217

TM-64 Weiss, R.B.
Finding Isomorph Classes for Combinatorial Structures, S.M. Thesis, EE Department, June 1975

TM-65 Fischer, M.J.
The Complexity Negation-Limited Networks - A Brief Survey, June 1975

TM-66 Leung, C.
Formal Properties of Well-Formed Data Flow Schemas, S.B., S.M. & E.E. Thesis, EE Department, June 1975

TM-67 Cardoza, E.E.
Computational Complexity of the Word Problem for Commutative Semigroups, S.M. Thesis, EE & CS Department, October 1975

TM-68 Weng, K-S.

	Stream-Oriented Computation in Recursive Data Flow Schemas, S.M. Thesis, EE & CS Department, October 1975
TM-69	Bayer, P.J. Improved Bounds on the Costs of Optimal and Balanced Binary Search Trees, S.M. Thesis, EE & CS Department, November 1975
TM-70	Ruth, G.R. Automatic Design of Data Processing Systems, February 1976; AD A023-451
TM-71	Rivest, R. On the Worst-Case of Behavior of String-Searching Algorithms, April 1976
TM-72	Ruth, G.R. Protosystem I: An Automatic Programming System Prototype, July 1976; AD A026-912
TM-73	Rivest, R. Optimal Arrangement of Keys in a Hash Table, July 1976
TM-74	Malvania, N. The Design of a Modular Laboratory for Control Robotics, S.M. Thesis, EE & CS Department, September 1976; AD A030-418
TM-75	Yao, A.C. and Rivest, R.I. K+1 Heads are Better than K, September 1976; AD A030-008
TM-76	Bloniarz, P.A., Fischer, M.J. and Meyer, A.R. A Note on the Average Time to Compute Transitive Closures, September 1976
TM-77	Mok, A.K. Task Scheduling in the Control Robotics Environment, S.M. Thesis, EE & CS Department, September 1976; AD A030-402
TM-78	Benjamin, A.J. Improving Information Storage Reliability Using a Data Network, S.M. Thesis, EE & CS Department, October 1976; AD A033-394
TM-79	Brown, G.P. A System to Process Dialogue: A Progress Report, October 1976; AD A033-276
TM-80	Even, S. The Max Flow Algorithm of Dinic and Karzanov: An Exposition, December 1976
TM-81	Gifford, D.K. Hardware Estimation of a Process' Primary Memory Requirements, S.B. Thesis, EE & CS Department, January 1977
TM-82	Rivest, R.L., Shamir, A. and Adelman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, April 1977; AD A039-036

TM-83	Baratz, A.E. Construction and Analysis of Network Flow Problem which Forces Karzanov Algorithm to $O(n^3)$ Running Time, April 1977
TM-84	Rivest, R.L. and Pratt, V.R. The Mutual Exclusion Problem for Unreliable Processes, April 1977
TM-85	Shamir, A. Finding Minimum Cutsets in Reducible Graphs, June 1977; AD A040-698
TM-86	Szolovits, P., Hawkinson, L.B. and Martin, W.A. An Overview of OWL, A Language for Knowledge Representation, June 1977; AD A041-372
TM-87	Clark, D., editor Ancillary Reports: Kernel Design Project, June 1977
TM-88	Lloyd, E.L. On Triangulations of a Set of Points in the Plane, S.M. Thesis, EE & CS Department, July 1977
TM-89	Rodriguez, H. Jr. Measuring User Characteristics on the Multics System, S.B. Thesis, EE & CS Department, August 1977
TM-90	d'Oliveira, C.R. An Analysis of Computer Decentralization, S.B. Thesis, EE & CS Department, October 1977; AD A045-526
TM-91	Shamir, A. Factoring Numbers in O(log n) Arithmetic Steps, November 1977; AD A047-709
TM-92	Misunas, D.P. Report on the Workshop on Data Flow Computer and Program Organization, November 1977
TM-93	Amikura, K. A Logic Design for the Cell Block of a Data-Flow Processor, S.M. Thesis, EE & CS Department, December 1977
TM-94	Berez, J.M. A Dynamic Debugging System for MDL, S.B. Thesis, EE & CS Department, January 1978; AD A050-191
TM-95	Harel, D. Characterizing Second Order Logic with First Order Quantifiers, February 1978
TM-96	Harel, D., Amir P. and Stavi, J. A Complete Axiomatic System for Proving Deductions about Recursive Programs, February 1978
TM-97	Harel, D., Meyer, A.R. and Pratt, V.R. Computability and Completeness in Logics of Programs, February 1978

TM-98	Harel, D. and Pratt, V.R. Nondeterminism in Logics of Programs, February 1978
TM-99	LaPaugh, A.S. The Subgraph Homeomorphism Problem, S.M. Thesis, EE & CS Department, February 1978
TM-100	Misunas, D.P. A Computer Architecture for Data-Flow Computation, S.M. Thesis, EE & CS Department, March 1978; AD A052-538
TM-101	Martin, W.A. Descriptions and the Specialization of Concepts, March 1978; AD A052-773
TM-102	Abelson, H. Lower Bounds on Information Transfer in Distributed Computations, April 1978
TM-103	Harel, D. Arithmetical Completeness in Logics of Programs, April 1978
TM-104	Jaffe, J. The Use of Queues in the Parallel Data Flow Evaluation of "If-Then-While" Programs, May 1978
TM-105	Masek, W.J. and Paterson, M.S. A Faster Algorithm Computing String Edit Distances, May 1978
TM-106	Parikh, R. A Completeness Result for a Propositional Dynamic Logic, July 1978
TM-107	Shamir, A. A Fast Signature Scheme, July 1978; AD A057-152
TM-108	Baratz, A.E. An Analysis of the Solovay and Strassen Test for Primality, July 1978
TM-109	Parikh, R. Effectiveness, July 1978
TM-110	Jaffe, J.M. An Analysis of Preemptive Multiprocessor Job Scheduling, September 1978
TM-111	Jaffe, J.M. Bounds on the Scheduling of Typed Task Systems, September 1978
TM-112	Parikh, R. A Decidability Result for a Second Order Process Logic, September 1978
TM-113	Pratt, V.R. A Near-optimal Method for Reasoning about Action, September 1978

TM-114 Dennis, J.B., Fuller, S.H., Ackerman, W.B., Swan, R.J. and Weng, K-S.
Research Directions in Computer Architecture, September 1978; AD A061-222

TM-115 Bryant, R.E. and Dennis, J.B.
Concurrent Programming, October 1978; AD A061-180

TM-116 Pratt, V.R.
Applications of Modal Logic to Programming, December 1978

TM-117 Pratt, V.R.
Six Lectures on Dynamic Logic, December 1978

TM-118 Borkin, S.A.
Data Model Equivalence, December 1978; AD A062-753

TM-119 Shamir, A. and Zippel, R.E.
On the Security of the Merkle-Hellman Cryptographic Scheme, December 1978; AD A063-104

TM-120 Brock, J.D.
Operational Semantics of a Data Flow Language, S.M. Thesis, EE & CS Department, December 1978; AD A062-997

TM-121 Jaffe, J.
The Equivalence of R.E. Programs and Data Flow Schemes, January 1979

TM-122 Jaffe, J.
Efficient Scheduling of Tasks Without Full Use of Processor Resources, January 1979

TM-123 Perry, H.M.
An Improved Proof of the Rabin-Hartmanis-Stearns Conjecture, S.M. & E.E. Thesis, EE & CS Department, January 1979

TM-124 Toffoli, T.
Bicontinuous Extensions of Invertible Combinatorial Functions, January 1979; AD A063-886

TM-125 Shamir, A., Rivest, R.L. and Adelman, L.M.
Mental Poker, February 1979; AD A066-331

TM-126 Meyer, A.R. and Paterson, M.S.
With What Frequency Are Apparently Intractable Problems Difficult?, February 1979

TM-127 Strazdas, R.J.
A Network Traffic Generator for Decnet, S.B. & S.M. Thesis, EE & CS Department, March 1979

TM-128 Loui, M.C.
Minimum Register Allocation is Complete in Polynomial Space, March 1979

TM-129 Shamir, A.

On the Cryptocomplexity of Knapsack Systems, April 1979; AD A067-972

TM-130 Greif, I. and Meyer, A.R.
Specifying the Semantics of While-Programs: A Tutorial and Critique of a Paper by Hoare and Lauer, April 1979; AD A068-967

TM-131 Adelman, L.M.
Time, Space and Randomness, April 1979

TM-132 Patil, R.S.
Design of a Program for Expert Diagnosis of Acid Base and Electrolyte Disturbances, May 1979

TM-133 Loui, M.C.
The Space Complexity of Two Pebble Games on Trees, May 1979

TM-134 Shamir, A.
How to Share a Secret, May 1979; AD A069-397

TM-135 Wyleczuk, R.H.
Timestamps and Capability-Based Protection in a Distributed Computer Facility, S.B. & S.M. Thesis, EE & CS Department, June 1979

TM-136 Misunas, D.P.
Report on the Second Workshop on Data Flow Computer and Program Organization, June 1979

TM-137 Davis, E. and Jaffe, J.M.
Algorithms for Scheduling Tasks on Unrelated Processors, June 1979

TM-138 Pratt, V.R.
Dynamic Algebras: Examples, Constructions, Applications, July 1979

TM-139 Martin, W.A.
Roles, Co-Descriptors, and the Formal Representation of Quantified English Expressions (Revised May 1980), September 1979; AD A074-625

TM-140 Szolovits, P.
Artificial Intelligence and Clinical Problem Solving, September 1979

TM-141 Hammer, M. and McLeod, D.
On Database Management System Architecture, October 1979; AD A076-417

TM-142 Lipski, W., Jr.
On Data Bases with Incomplete Information, October 1979

TM-143 Leth, J.W.
An Intermediate Form for Data Flow Programs, S.M. Thesis, EE & CS Department, November 1979

TM-144 Takagi, A.

Concurrent and Reliable Updates of Distributed Databases, November 1979

TM-145 Loui, M.C.
A Space Bound for One-Tape Multidimensional Turing Machines, November 1979

TM-146 Aoki, D.J.
A Machine Language Instruction Set for a Data Flow Processor, S.M. Thesis, EE & CS Department, December 1979

TM-147 Schroeppel, R. and Shamir, A.
A $T = 0(2^{n/2})$, $S = 0(2^{n/4})$ Algorithm for Certain NP-Complete Problems, January 1980; AD A080-385

TM-148 Adelman, L.M. and Loui, M.C.
Space-Bounded Simulation of Multitape Turing Machines, January 1980

TM-149 Pallottino, S. and Toffoli, T.
An Efficient Algorithm for Determining the Length of the Longest Dead Path in an "Lifo" Branch-and-Bound Exploration Schema, January 1980; AD A079-912

TM-150 Meyer, A.R.
Ten Thousand and One Logics of Programming, February 1980

TM-151 Toffoli, T.
Reversible Computing, February 1980; AD A082-021

TM-152 Papadimitriou, C.H.
On the Complexity of Integer Programming, February 1980

TM-153 Papadimitriou, C.H.
Worst-Case and Probabilistic Analysis of a Geometric Location Problem, February 1980

TM-154 Karp, R.M. and Papadimitriou, C.H.
On Linear Characterizations of Combinatorial Optimization Problems, February 1980

TM-155 Atai, A., Lipton, R.J., Papadimitriou, C.H. and Rodeh, M.
Covering Graphs by Simple Circuits, February 1980

TM-156 Meyer, A.R. and Parikh, R.
Definability in Dynamic Logic, February 1980

TM-157 Meyer, A.R. and Winklmann, K.
On the Expressive Power of Dynamic Logic, February 1980

TM-158 Stark, E.W.
Semaphore Primitives and Starvation-Free Mutual Exclusion, S.M. Thesis, EE & CS Department, March 1980

TM-159 Pratt, V.R.
Dynamic Algebras and the Nature of Induction, March 1980

TM-160 Kanellakis, P.C.

	On the Computational Complexity of Cardinality Constraints in Relational Databases, March 1980
TM-161	Lloyd, E.L. Critical Path Scheduling of Task Systems with Resource and Processor Constraints, March 1980
TM-162	Marcum, A.M. A Manager for Named, Permanent Objects, S.B. & S.M. Thesis, EE & CS Department, April 1980; AD A083-491
TM-163	Meyer, A.R. and Halpern, J.Y. Axiomatic Definitions of Programming Languages: A Theoretical Assessment, April 1980
TM-164	Shamir, A. The Cryptographic Security of Compact Knapsacks (Preliminary Report), April 1980; AD A084-456
TM-165	Finseth, C.A. Theory and Practice of Text Editors or A Cookbook for an Emacs, S.B. Thesis, EE & CS Department, May 1980
TM-166	Bryant, R.E. Report on the Workshop on Self-Timed Systems, May 1980
TM-167	Pavelle, R. and Wester, M. Computer Programs for Research in Gravitation and Differential Geometry, June 1980
TM-168	Greif, I. Programs for Distributed Computing: The Calendar Application, July 1980; AD A087-357
TM-169	Burke, G. and Moon, D. LOOP Iteration Macro, (revised January 198l) July 1980; AD A087-372
TM-170	Ehrenfeucht, A., Parikh, R. and Rozenberg, G. Pumping Lemmas for Regular Sets, August 1980
TM-171	Meyer, A.R. What is a Model of the Lambda Calculus?, August 1980
TM-172	Paseman, W.G. Some New Methods of Music Synthesis, S.M. Thesis, EE & CS Department, August 1980; AD A090-130
TM-173	Hawkinson, L.B. XLMS: A Linguistic Memory System, September 1980; AD A090-033
TM-174	Arvind, Kathail, V. and Pingali, K. A Dataflow Architecture with Tagged Tokens, Sep' :mber 1980
TM-175	Meyer, A.R., Weise, D. and Loui, M.C. On Time Versus Space III, September 1980

TM-176	Seaquist, C.R. A Semantics of Synchronization, S.M. Thesis, EE & CS Department, September 1980; AD A091-015
TM-177	Sinha, M.K. TIMEPAD - A Performance Improving Synchronization Mechanism for Distributed Systems, September 1980
TM-178	Arvind and Thomas, R.E. I-Structures: An Efficient Data Type for Functional Languages, September 1980
TM-179	Halpern, J.Y. and Meyer, A.R. Axiomatic Definitions of Programming Languages, II, October 1980
TM-180	Papadimitriou, C.H. A Theorem in Database Concurrency Control, October 1980
TM-181	Lipski, W. Jr. and Papadimitriou, C.H. A Fast Algorithm for Testing for Safety and Detecting Deadlocks in Locked Transaction Systems, October 1980
TM-182	Itai, A., Papadimitriou, C.H. and Szwarefiter, J.L. Hamilton Paths in Grid Graphs, October 1980
TM-183	Meyer, A.R. A Note on the Length of Craig's Interpolants, October 1980
TM-184	Lieberman, H. and Hewitt, C. A Real Time Garbage Collector that can Recover Temporary Storage Quickly, October 1980
TM-185	Kung, H-T. and Papadimitriou, C.H. An Optimality Theory of Concurrency Control for Databases, November 1980; AD A092-625
TM-186	Szolovits, P. and Martin, W.A. BRAND X Manual, November 1980; AD A093-041
TM-187	Fischer, M.J., Meyer, A.R. and Paterson, M.S. $\Omega(n \log n)$ Lower Bounds on Length of Boolean Formulas, November 1980
TM-188	Mayr, E. An Effective Representation of the Reachability Set of Persistent Petri Nets, January 1981
TM-189	Mayr, E. Persistence of Vector Replacement Systems is Decidable, January 1981
TM-190	Ben-Ari, M., Halpern, J.Y. and Pnueli, A. Deterministic Propositional Dynamic Logic: Finite Models, Complexity, and Completeness, January 1981.
TM-191	Parikh, R. Propositional Dynamic Logics of Programs: A Survey, January 1981.

TM-192	Meyer, A.R., Streett, R.S. and Mirkowska, G. The Deducibility Problem in Propositional Dynamic Logic, February 1981
TM-193	Yannakakis, M. and Papadimitriou, C.H. Algebraic Dependencies, February 1981
TM-194	Barendregt, H. and Longo, G. Recursion Theoretic Operators and Morphisms on Numbered Sets, February 1981
TM-195	Barber, G.R. Record of the Workshop on Research in Office Semantics, February 1981
TM-196	Bhatt, S.N. On Concentration and Connection Networks, S.M. Thesis, EE & CS Department, March 1981
TM-197	Fredkin, E. and Toffoli, T. Conservative Logic, May 1981
TM-198	Halpern, J. and Reif, J. The Propositional Dynamic Logic of Deterministic Well-Structured Programs, March 1981
TM-199	Mayr, E. and Meyer, A.R. The Complexity of the Word Problems for Communative Semigroups and Polynomial Ideals, June 1981
TM-200	Burke, G.S. LSB Manual, June 1981
TM-201	Meyer, A.R. What is a Model of the lambda Calculus? Expanded Version, July 1981.
TM-202	Saltzer, J.H. Communication Ring Initialization without Central Control, December 1981
TM-203	Bawden, A., Burke, G. and Hoffman, C. Maclisp Extensions, July 1981
TM-204	Halpern, J.Y. On the Expressive Power of Dynamic Logic, II, August 1981
TM-205	Kannon, R. Circuit-Size Lower Bounds and Non-Reducibility to Sparce Sets, October 1981.
TM-206	Leiserson, C. and Pinter, R. Optimal Placement for River Routing, October 1981
TM-207	Longo, G. Power Set Models For Lambda-Calculus: Theories, Expansions, Isomorphisms, November 1981

TM-208	Cosmadakis, S. and Papadimitriou, C. The Traveling Salesman Problem with Many Visits to Few Cities, November 1981
TM-209	Johnson, D. and Papadimitriou, C. Computational Complexity and the Traveling Salesman Problem, December 1981
TM-210	Greif, I. Software for the 'Roiels' People Play, February 1982
TM-211	Meyer, A.R. and Tiuryn, J. A Note on Equivalences Among Logics of Programs, December 1981
TM-212	Elias, P. Minimax Optimal Universal Codeword Sets, January 1982
TM-213	Greif, I. PCAL: A Personal Calendar, January 1982
TM-214	Meyer, A. and Mitchell, J. Terminations for Recursive Programs: Completeness and Axiomatic Definability, March 1982
TM-215	Leiserson, C. and Saxe J. Optimizing Synchronous Systems, March 1982
TM-216	Church, K. and Patil, R. Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table, April 1982.
TM-217	Wright, D. A File Transfer Program for a Personal Computer, April 1982
TM-218	Greif, I. Cooperative Office Work, Teleconferencing and Calendar Management: A Collection of Papers, May 1982
TM-219	Jouannaud, J-P., Lescanne, P. and Reinig, F. Recursive Decomposition Ordering and Multiset Orderings, June 1982
TM-220	Chu, T-A. Circuit Analysis of Self-Times Elements for NMOS VLSI Systems, May 1982
TM-221	Leighton, F., Lepley, M. and Miller, G. Layouts for the Shuffle-Exchange Graph Based on the Complex Plane Diagram, June 1982
TM-222	Meier zu Sieker, F. A Telex Gateway for the Internet, S.B. Thesis, Electrical Engineering Department, May 1982
TM-223	diSessa, A.A. A Principled Design for an Integrated Computation Environment, July 1982

TM-224	Barber, G. Supporting Organizational Problem Solving with a Workstation, July 1982
TM-225	Barber, G. and Hewitt, C. Foundations for Office Semantics, July 1982
TM-226	Bergstra, J., Chmielinska, A. and Tiuryn, J. Hoares's Logic Not Complete When it Could Be, August 1982
TM-227	Leighton, F.T. New Lower Bound Techniques for VLSI, August 1982
TM-228	Papadimitriou, C. and Zachos, S. Two Remarks on the Power of Counting, August 1982
TM-229	Cosmadakis, S. The Complexity of Evaluation Relational Queries, August 1982
TM-230	Shamir, A. Embedding Cryptographic Trapdoors in Arbitrary Knapsack Systems, September 1982
TM-231	Kleitman, D., Leighton, F.T., Lepley, M. and Miller G. An Asymptotically Optimal Layout for the shuffle-exchange Graph, October 1982
TM-232	Yeh, A. PLY: A System of Plausibility Inference with a Probabilistic Basis, December 1982
TM-233	Konopelski, L. Implementing Internet Remote Logi on a Personal Computer, S.B. Thesis, Electrical Engineering Department, December 1982
TM-234	Rivest, R. and Sherman, A. Randomized Encryption Techniques, January 1983.
TM-235	Mitchell, J. The Implication of Problem for Functional and Inclusion Dependencies, February 1983
TM-236	Leighton, F.T. and Leiserson, C.E. Wafter-Scale Integration of Systolic Arrays, February 1983
TM-237	Dolev, D., Leighton, F.T. and Trickey, H. Planar Embedding of Planar Graphs, February 1983
TM-238	Baker, B.S., Bhatt, S.N. and Leighton, F.T. An Approximation Algorithm for Manhattan Routing, February 1983
TM-239	Sutherland, J.B. and Sirbu, M. Evaluation of an Office Analysis Methodology, March 1983
TM-240	Bromley, H. A Program for Therapy of Acid-Base and Electrolyte Disorders, S.B. Thesis, Electrical Engineering Department, June 1983
TM-241	Arvind and Iannucci, R.A.

	Two Fundamental Issues in Multiprocessing: The Dataflow Solution, September 1983
TM-242	Pingali, K. and Arvind. Efficient Demand-driven Evaluation (I), September 1983
TM-243	Pingali, K. and Arvind. Efficient Demand-driven Evaluation (II), September 1983
TM-244	Goldreich, O., Goldwasser, S. and Micali, S. How to Construct Random Functions, November 1983
TM-245	Meyer, A. Understanding Algol: The View of the Recent Convert to Denotational Semantics, October 1983
TM-246	Trakhtenbrot, B.A., Halpern, J.Y. and Meyer, A.R. From Denotational to Operational and Axiomatic Semantics for Algol-Like Languages: An Overview, October 1983
TM-247	Leighton, T. and Lepley, M. Probabilistic Searching in Sorted Linked Lists, November 1983
TM-248	Leighton, F.T. and Rivest, R.L. Estimating a Probability Using Finite Memory, November 1983
TM-249	Leighton, F.T. and Rivest, R.L. The Markov Chain Tree Theorem, December 1983
TM-250	Goldreich, O. On Concurrent Identification Protocols, December 1983
TM-251	Dolev, D., Lynch, N., Pinter, S. Stark, E. and Weihl, W. Reaching Approximate Agreement in the Presence of Faults, December 1983
TM-252	Zachos, S. and Heller, H. On BPP, December 1983
TM-253	Chor, B., Leiserson, C., Rivest, R. and Shearer, J. An Application of Number Theory to the Organization of Raster Graphics Memory, April 1984
TM-254	Feldmeier, D.C. Empirical Analysis of a Token Ring Network, April 1984
TM-255	Bhatt, S. and Leiserson, C. How to Assemble Tree Machines, April 1984
TM-256	Goldreich, O. On the Number of Close-and Equal Pairs of Bits in a String (With Implications on the Security of RSA's L.S.B., April 1984
TM-257	Dwork, C., Kanellakis, P. and Mitchell, J. On the Sequential Nature of Unification, April 1984
TM-258	Halpern, M., Meyer A. and Trakhtenbrot, B. The Semantics of Local Storage, or What Makes the Free-list Free?, April 1984

TM-259	Lynch, N. and Fredrickson, G. The Impact of Synchronous Communication on the Problem of Electing a Leader in a Ring, April 1984
TM-260	Chor, B. and Goldreich, O. RSA/Rabin Least Significant Bits are 1/2 + poly(logn) Secure, May 1984
TM-261	Zaks, S. Optimal Distributed Algorithms for Sorting and Ranking, May 1984
TM-262	Leighton, T. and Rosenberg, A. Three-dimensional Circuit Layouts, June 1984
TM-263	Sirbu, M.S. and Sutherland, J.B. Naming and Directory Issues in Message Transfer Systems, July 1984
TM-264	Sarin, S.K. and Greif, I. Software for Interactive On-Line Conferences, July 1984
TM-265	Lundelius, J. and Lynch, N. A New Fault-Tolerant Algorithm for Clock Synchronization, July 1984
TM-266	Chor, B. and Coan, B.A. A Simple and Efficient Randomized Byzantine Agreement Algorithm, August 1984
TM-267	Schooler, R. and Stamos, J.W. Proposal for a Small Scheme Implementation, October 1984
TM-268	Awerbuch, B. Complexity of Network Synchronization, January 1985
TM-269	Fisher, M., Lynch, N.A., Burns, J. and Borodin, A. The Colored Ticket Algorithm, August 1983
TM-270	Dwork, C., Lynch, C. and Stockmeyer, L. Consensus in the Presence of Partial Synchrony (Preliminary Version), July 1984
TM-271	Dershowitz, N. and Zaks, S. Patterns in Trees, January 1985
TM-272	Leighton, T. Tight Bounds on the Complexity of Parallel Sorting, April 1985
TM-273	Berman, F., Leighton, T., Shor, P.W. and Shor, L. Generalized Planar Matching, April 1985
TM-274	Kuipers, B. Qualitative Simulation of Mechanisms, April 1985
TM-275	Burns, J.E. and Lynch, N.A. The Byzantine Firing Squad Problem, April 1985
TM-276	Dolev, D., Lynch., N.A., Pinter, S.S., Stark, E.W. and Weihl, W.E. Reaching Approximate Agreement in the Presence of Faults, May 1985

TM-277	Frederickson, G.N. and Lynch, N.A. A General Lower Bound for Electing a Leader in a Ring, March 1985
TM-278	Fisch, M.J., Griffeth, N.D., Guibas, L.J. and Lynch, N.A. Probabilistic Analysis of a Network Resource Allocation Algorithm, June 1985
TM-279	Fischer, M.J., Lynch, N.A. and Merritt, M. Easy Impossibility Proofs for Distributed Consensus Problems, June 1985
TM-280	Kuipers, B. and Kassirer, J.P. Qualitative Simulation in Medical Physiology: A Progress Report, June 1985
TM-281	Hailperin, M. What Price for Eliminating Expression Side-Effects?, June 1985
TM-285	Kilian, J.J Two Undecidability Results in Probabilistic Automata Theory, S.B. Thesis/June 1985
TM-288	Chung, J.C. Dscribe: A Scribe Server, May 1985
TM-290	Fisher, M.J., Lynch, N.A., Burns, J.E. and Borcdin, A. Distributed FIFO Allocation of Identical Resources Using Small Shared Space, June 1985
TM-291	Goldberg, A.V. A New Max-Flow Algorithm, November 1985
TM-292	Jain, R. and Routhier, S. Packet Trains: Measurements and a New Model for Computer Network Traffic, November 1985
TM-293	Barrington, D.A. Width-3 Permutation Branching Programs, December 1985
TM-294	Arvind and Culler, D.E. Dataflow Architectures, February 1986
TM-295	Greif, I., Selinger, R. and Weihl, W. Atomic Data Abstractions in a Distributed Collaborative Editing System (Extended Abstract), November 1985
TM-296	Margolus, N., Toffoli, T. and Vichniac, G. Cellular Automata Supercomputers for Fluid Dynamics Modeling, December 1985
TM-297	Bentley, J.L., Leighton, F.T., Lepley, M., Stanat, D.F. and Steele, J.M. A Randomized Data Structure for Ordered Sets, May 1986
TM-298	Leighton, T. and Shor, P. Tight Bounds for Minimax Grid Matching, With Applications to the Average Case Analysis of Algorithms, May 1986

TM-299 Gifford, D.K., Lucassen, J.M. and Berlin, S.T,
The Application of Digital Broadcast Communication to Large Scale Information Systems, April 1986

TM-300 Dwork, C. and Moses, Y.
Knowledge and Common Knowledge in a Byzantine Environment: Crash Failures, July 1986

TM-301 Elias, P.
Interval and Recency-Rank Source Coding: Two On-Line Adaptive Variable-Length Schemes, April 1986

TM-302 Leighton, T. and Leiserson, C.E.
A Survey of Algorithms for Integrating Wafer-Scale Systolic Arrays, May 1986

TM-303 Li, M., Longpre, L. and Vitanyi, P.M.B.
The Power of the Queue, April 1986

TM-304 Kranakis, E. and Vitanyi, P.M.B.
Distributed Control in Computer Networks and Cross-Sections of Colored Multidimensional Bodies, April 1986

TM-305 Sacks, E.
Representing Change, May 1986

TM-306 Vitanyi, P.M.B.
Nonsequential Computation and Laws of Nature, May 1986

TM-307 Greenberg, R.I. and Leiserson, C.E.
Randomized Routing on Fat-Trees, April 1986

TM-308 Meyer, A.R.
Floyd-Hoare Logic Defines Semantics, May 1986

TM-309 Leiserson, C.E. and Saxe, J.B.
Retiming Synchronous Circuitry, May 1986

Technical Reports

TR-1[6]

Bobrow, D.G.
Natural Language Input for a Computer Problem Solving System, Ph.D. Dissertation, Math. Department, September 1964; AD 604-730

TR-2 Raphael, B.
SIR: A Computer Program for Semantic Information Retrieval, Ph.D. Dissertation, Math. Department, June 1964; AD 608-499

[6]TRs 5, 9, 10, 15 were never issued

TR-3	Corbato, F.J. System Requirements for Multiple-Access, Time-Shared Computers, May 1964; AD 608-501
TR-4	Ross, D.T. and Feldman, C.G. Verbal and Graphical Language for the AED System: A Progress Report, May 1964; AD 604-678
TR-6	Biggs, J.M. and Logcher, R.D. STRESS: A Problem-Oriented Language for Structural Engineering, May 1964; AD 604-679
TR-7	Weizenbaum, J. OPL-1: An Open Ended Programming System within CTSS, April 1964; AD 604-680
TR-8	Greenberger, M. The OPS-1 Manual, May 1964; AD 604-681
TR-11	Dennis, J.B. Program Structure in a Multi-Access Computer, May 1964; AD 608-500
TR-12	Fano, R.M. The MAC System: A Progress Report, October 1964; AD 609-296
TR-13	Greenberger, M. A New Methodology for Computer Simulation, October 1964; AD 609-288
TR-14	Roos, D. Use of CTSS in a Teaching Environment, November 1964; AD 661-807
TR-16	Saltzer, J.H. CTSS Technical Notes, March 1965; AD 612-702
TR-17	Samuel, A.L. Time-Sharing on a Multiconsole Computer, March 1965; AD 462-158
TR-18	Scherr, A.L. An Analysis of Time-Shared Computer Systems, Ph.D. Dissertation, EE Department, June 1965; AD 470-715
TR-19	Russo, F.J. A Heuristic Approach to Alternate Routing in a Job Shop, S.B. & S.M. Thesis, Sloan School, June 1965; AD 474-018
TR-20	Wantman, M.E. CALCULAID: An On-Line System for Algebraic Computation and Analysis, S.M. Thesis, Sloan School, September 1965; AD 474-019
TR-21	Denning, P.J. Queueing Models for File Memory Operation, S.M. Thesis, EE Department, October 1965; AD 624-943

TR-22 Greenberger, M.
The Priority Problem, November 1965; AD 625-728

TR-23 Dennis, J.B. and Van Horn, E.C.
Programming Semantics for Multi-programmed Computations, December 1965; AD 627-537

TR-24 Kaplow, R., Strong, S. and Brackett, J.
MAP: A System for On-Line Mathematical Analysis, January 1966; AD 476-443

TR-25 Stratton, W.D.
Investigation of an Analog Technique to Decrease Pen-Tracking Time in Computer Displays, S.M. Thesis, EE Department, March 1966; AD 631-396

TR-26 Cheek, T.B.
Design of a Low-Cost Character Generator for Remote Computer Displays, S.M. Thesis, EE Department, March 1966; AD 631-269

TR-27 Edwards, D.J.
OCAS - On-Line Cryptanalytic Aid System, S.M. Thesis, EE Department, May 1966; AD 633-678

TR-28 Smith, A.A.
Input/Output in Time-Shared, Segmented, Multiprocessor Systems, S.M. Thesis, EE Department, June 1966; AD 637-215

TR-29 Ivie, E.L.
Search Procedures Based on Measures of Relatedness between Documents, Ph.D. Dissertation, EE Department, June 1966; AD 636-275

TR-30 Saltzer, J.H. Traffic Control in a Multiplexed Computer System, Sc.D. Thesis, EE Department, July 1966; AD 635-966

TR-31 Smith, D.L.
Models and Data Structures for Digital Logic Simulation, S.M. Thesis, EE Department, August 1966; AD 637-192

TR-32 Teitelman, W.
PILOT: A Step Toward Man-Computer Symbiosis, Ph.D. Dissertation, Math. Department, September 1966; AD 638-446

TR-33 Norton, L.M. ADEPT - A Heuristic Program for Proving Theorems of Group Theory, Ph.D. Dissertation, Math. Department, October 1966; AD 645-660

TR-34 Van Horn, E.C., Jr.
Computer Design for Asynchronously Reproducible Multiprocessing, Ph.D. Dissertation, EE Department, November 1966; AD 650-407

TR-35 Fenichel, R.R.
An On-Line System for Algebraic Manipulation, Ph.D. Dissertation, Appl. Math. (Harvard), December 1966; AD 657-282

TR-36	Martin, W.A. Symbolic Mathematical Laboratory, Ph.D. Dissertation, EE Department, January 1967; AD 657-283
TR-37	Guzman-Arenas, A. Some Aspects of Pattern Recognition by Computer, S.M. Thesis, EE Department, February 1967; AD 656-041
TR-38	Rosenberg, R.C., Kennedy, D.W. and Humphrey, R.A. A Low-Cost Output Terminal For Time-Shared Computers, March 1967; AD 662-027
TR-39	Forte, A. Syntax-Based Analytic Reading of Musical Scores, April 1967; AD 661-806
TR-40	Miller, J.R. On-Line Analysis for Social Scientists, May 1967; AD 668-009
TR-41	Coons, S.A. Surfaces for Computer-Aided Design of Space Forms, June 1967; AD 663-504
TR-42	Liu, C.L., Chang, G.D. and Marks, R.E. Design and Implementation of a Table-Driven Compiler System, July 1967; AD 668-960
TR-43	Wilde, D.U. Program Analysis by Digital Computer, Ph.D. Dissertation, EE Department, August 1967; AD 662-224
TR-44	Gorry, G.A. A System for Computer-Aided Diagnosis, Ph.D. Dissertation, Sloan School, September 1967; AD 662-665
TR-45	Leal-Cantu, N. On the Simulation of Dynamic Systems with Lumped Parameters and Time Delays, S.M. Thesis, ME Department, October 1967; AD 663-502
TR-46	Alsop, J.W. A Canonic Translator, S.B. Thesis, EE Department, November 1967; AD 663-503
TR-47	Moses, J. Symbolic Integration, Ph.D. Dissertation, Math. Department, December 1967; AD 662-666
TR-48	Jones, M.M. Incremental Simulation on a Time-Shared Computer, Ph.D. Dissertation, Sloan School, January 1968; AD 662-225
TR-49	Luconi, F.L. Asynchronous Computational Structures, Ph.D. Dissertation, EE Department, February 1968; AD 667-602
TR-50	Denning, P.J.

Resource Allocation in Multiprocess Computer Systems, Ph.D. Dissertation, EE Department, May 1968; AD 675-554

TR-51 Charniak, E.
CARPS, A Program which Solves Calculus Word Problems, S.M. Thesis, EE Department, July 1968; AD 673-670

TR-52 Deitel, H.M.
Absentee Computations in a Multiple-Access Computer System, S.M. Thesis, EE Department, August 1968; AD 684-738

TR-53 Slutz, D.R.
The Flow Graph Schemata Model of Parallel Computation, Ph.D. Dissertation, EE Department, September 1968; AD 683-393

TR-54 Grochow, J.M.
The Graphic Display as an Aid in the Monitoring of a Time-Shared Computer System, S.M. Thesis, EE Department, October 1968; AD 689-468

TR-55 Rappaport, R.L.
Implementing Multi-Process Primitives in a Multiplexed Computer System, S.M. Thesis, EE Department, November 1968; AD 689-469

TR-56 Thornhill, D.E., Stotz, R.H., Ross, D.T. and Ward, J.E.
An Integrated Hardware-Software System for Computer Graphics in Time-Sharing, December 1968; AD 685-202

TR-57 Morris, J.H.
Lambda-Calculus Models of Programming Languages, Ph.D. Dissertation, Sloan School, December 1968; AD 683-394

TR-58 Greenbaum, H.J.
A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System, S.M. Thesis, EE Department, January 1969; AD 686-988

TR-59 Guzman; A.
Computer Recognition of Three- Dimensional Objects in a Visual Scene, Ph.D. Dissertation, EE Department, December 1968; AD 692-200

TR-60 Ledgard, H.F.
A Formal System for Defining the Syntax and Semantics of Computer Languages, Ph.D. Dissertation, EE Department, April 1969; AD 689-305

TR-61 Baecker, R.M.
Interactive Computer-Mediated Animation, Ph.D. Dissertation, EE Department, June 1969; AD 690-887

TR-62 Tillman, C.C., Jr.
EPS: An Interactive System for Solving Elliptic Boundary-Value Problems with Facilities for Data Manipulation and General-Purpose Computation, June 1969; AD 692-462

TR-63 Brackett, J.W., Hammer, M. and Thornhill, D.E.
Case Study in Interactive Graphics Programming: A Circuit Drawing and Editing Program for Use with a Storage-Tube Display Terminal, October 1969; AD 699-930

TR-64 Rodriguez, J.E.
A Graph Model for Parallel Computations, Sc.D. Thesis, EE Department, September 1969; AD 697-759

TR-65 DeRemer, F.L.
Practical Translators for LR(k) Languages, Ph.D. Dissertation, EE Department, October 1969; AD 699-501

TR-66 Beyer, W.T.
Recognition of Topological Invariants by Iterative Arrays, Ph.D. Dissertation, Math. Department, October 1969; AD 699-502

TR-67 Vanderbilt, D.H.
Controlled Information Sharing in a Computer Utility, Ph.D. Dissertation, EE Department, October 1969; AD 699-503

TR-68 Selwyn, L.L.
Economies of Scale in Computer Use: Initial Tests and Implications for The Computer Utility, Ph.D. Dissertation, Sloan School, June 1970; AD 710-011

TR-69 Gertz, J.L.
Hierarchical Associative Memories for Parallel Computation, Ph.D. Dissertation, EE Department, June 1970; AD 711-091

TR-70 Fillat, A.I. and Kraning, L.A.
Generalized Organization of Large Data-Bases: A Set-Theoretic Approach to Relations, S.B. & S.M. Thesis, EE Department, June 1970; AD 711-060

TR-71 Fiasconaro, J.G.
A Computer-Controlled Graphical Display Processor, S.M. Thesis, EE Department, June 1970; AD 710-479

TR-72 Patil, S.S.
Coordination of Asynchronous Events, Sc.D. Thesis, EE Department, June 1970; AD 711-763

TR-73 Griffith, A.K.
Computer Recognition of Prismatic Solids, Ph.D. Dissertation, Math. Department, August 1970; AD 712-069

TR-74 Edelberg, M.
Integral Convex Polyhedra and an Approach to Integralization, Ph.D. Dissertation, EE Department, August 1970; AD 712-070

TR-75 Hebalkar, P.G.
Deadlock-Free Sharing of Resources in Asynchronous Systems, Sc.D. Thesis, EE Department, September 1970; AD 713-139

TR-76 Winston, P.H.

Learning Structural Descriptions from Examples, Ph.D. Dissertation, EE Department, September 1970; AD 713-988

TR-77 Haggerty, J.P.
Complexity Measures for Language Recognition by Canonic Systems, S.M. Thesis, EE Department, October 1970; AD 715-134

TR-78 Madnick, S.E.
Design Strategies for File Systems, S.M. Thesis, EE Department & Sloan School, October 1970; AD 714-269

TR-79 Horn, B.K.
Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View, Ph.D. Dissertation, EE Department, November 1970; AD 717-336

TR-80 Clark, D.D., Graham, R.M., Saltzer, J.H. and Schroeder, M.D.
The Classroom Information and Computing Service, January 1971; AD 717-857

TR-81 Banks, E.R.
Information Processing and Transmission in Cellular Automata, Ph.D. Dissertation, ME Department, January 1971; AD 717-951

TR-82 Krakauer, L.J.
Computer Analysis of Visual Properties of Curved Objects, Ph.D. Dissertation, EE Department, May 1971; AD 723-647

TR-83 Lewin, D.E.
In-Process Manufacturing Quality Control, Ph.D. Dissertation, Sloan School, January 1971; AD 720-098

TR-84 Winograd, T.
Procedures as a Representation for Data in a Computer Program for Understanding Natural Language, Ph.D. Dissertation, Math. Department, February 1971; AD 721-399

TR-85 Miller, P.L.
Automatic Creation of a Code Generator from a Machine Description, E.E. Thesis, EE Department, May 1971; AD 724-730

TR-86 Schell, R.R.
Dynamic Reconfiguration in a Modular Computer System, Ph.D. Dissertation, EE Department, June 1971; AD 725-859

TR-87 Thomas, R.H.
A Model for Process Representation and Synthesis, Ph.D. Dissertation, EE Department, June 1971; AD 726-049

TR-88 Welch, T.A.
Bounds on Information Retrieval Efficiency in Static File Structures, Ph.D. Dissertation, EE Department, June 1971; AD 725-429

TR-89 Owens, R.C., Jr.
Primary Access Control in Large-Scale Time-Shared Decision Systems, S.M. Thesis, Sloan School, July 1971; AD 728-036

TR-90 Lester, B.P.
Cost Analysis of Debugging Systems, S.B. & S.M. Thesis, EE Department, September 1971; AD 730-521

TR-91 Smoliar, S.W.
A Parallel Processing Model of Musical Structures, Ph.D. Dissertation, Math. Department, September 1971; AD 731-690

TR-92 Wang, P.S.
Evaluation of Definite Integrals by Symbolic Manipulation, Ph.D. Dissertation, Math. Department, October 1971; AD 732-005

TR-93 Greif, I.G.
Induction in Proofs about Programs, S.M. Thesis, EE Department, February 1972; AD 737-701

TR-94 Hack, M.H.T.
Analysis of Production Schemata by Petri Nets, S.M. Thesis, EE Department, February 1972; AD 740-320

TR-95 Fateman, R.J.
Essays in Algebraic Simplification (A revision of a Harvard Ph.D. Dissertation), April 1972; AD 740-132

TR-96 Manning, F.
Autonomous, Synchronous Counters Constructed Only of J-K Flip-Flops, S.M. Thesis, EE Department, May 1972; AD 744-030

TR-97 Vilfan, B.
The Complexity of Finite Functions, Ph.D. Dissertation, EE Department, March 1972; AD 739-678

TR-98 Stockmeyer, L.J.
Bounds on Polynomial Evaluation Algorithms, S.M. Thesis, EE Department, April 1972; AD 740-328

TR-99 Lynch, N.A.
Relativization of the Theory of Computational Complexity, Ph.D. Dissertation, Math. Department, June 1972; AD 744-032

TR-100 Mandl, R.
Further Results on Hierarchies of Canonic Systems, S.M. Thesis, EE Department, June 1972; AD 744-206

TR-101 Dennis, J.B.
On the Design and Specification of a Common Base Language, June 1972; AD 744-207

TR-102 Hossley, R.F.
Finite Tree Automata and Ω-Automata, S.M. Thesis, EE Department, September 1972; AD 749-367

TR-103 Sekino, A.
Performance Evaluation of Multiprogrammed Time-Shared Computer Systems, Ph.D. Dissertation, EE Department, September 1972; AD 749-949

TR-104 Schroeder, M.D.
Cooperation of Mutually Suspicious Subsystems in a Computer Utility, Ph.D. Dissertation, EE Department, September 1972; AD 750-173

TR-105 Smith, B.J.
An Analysis of Sorting Networks, Sc.D. Thesis, EE Department, October 1972; AD 751-614

TR-106 Rackoff, C.W.
The Emptiness and Complementation Problems for Automata on Infinite Trees, S.M. Thesis, EE Department, January 1973; AD 756<-248

TR-107 Madnick, S.E.
Storage Hierarchy Systems, Ph.D. Dissertation, EE Department, April 1973; AD 760-001

TR-108 Wand, M.
Mathematical Foundations of Formal Language Theory, Ph.D. Dissertation, Math. Department, December 1973.

TR-109 Johnson, D.S.
Near-Optimal Bin Packing Algorithms, Ph.D. Dissertation, Math. Department, June 1973, PB 222-090

TR-110 Moll, R.
Complexity Classes of Recursive Functions, Ph.D. Dissertation, Math. Department, June 1973; AD 767-730

TR-111 Linderman, J.P.
Productivity in Parallel Computation Schemata, Ph.D. Dissertation, EE Department, December 1973, PB 226-159/AS

TR-112 Hawryszkiewycz, I.T.
Semantics of Data Base Systems, Ph.D. Dissertation, EE Department, December 1973, PB 226-061/AS

TR-113 Herrmann, P.P.
On Reducibility Among Combinatorial Problems, S.M. Thesis, Math. Department, December 1973, PB 226-157/AS

TR-114 Metcalfe, R.M.
Packet Communication, Ph.D. Dissertation, Applied Math., Harvard University, December 1973; AD 771-430

TR-115 Rotenberg, L.
Making Computers Keep Secrets, Ph.D. Dissertation, EE Department, February 1974, PB 229-352/AS

TR-116 Stern, J.A.
Backup and Recovery of On-Line Information in a Computer Utility, S.M. & E.E. Thesis, EE Department, January 1974; AD 774-141

TR-117 Clark, D.D.
An Input/Output Architecture for Virtual Memory Computer

Systems, Ph.D. Dissertation, EE Department, January 1974; AD 774-738

TR-118 Briabrin, V.
An Abstract Model of a Research Institute: Simple Automatic Programming Approach, March 1974, PB 231-505/AS

TR-119 Hammer, M.M.
A New Grammatical Transformation into Deterministic Top-Down Form, Ph.D. Dissertation, EE Department, February 1974; AD 775-545

TR-120 Ramchandani, C.
Analysis of Asynchronous Concurrent Systems by Timed Petri Nets, Ph.D. Dissertation, EE Department, February 1974; AD 775-618

TR-121 Yao, F.F.
On Lower Bounds for Selection Problems, Ph.D. Dissertation, Math. Department, March 1974, PB 230-950/AS

TR-122 Scherf, J.A.
Computer and Data Security: A Comprehensive Annotated Bibliography, S.M. Thesis, Sloan School, January 1974; AD 775-546

TR-123 Introduction to Multics
February 1974; AD 918-562

TR-124 Laventhal, M.S.
Verification of Programs Operating on Structured Data, S.B. & S.M. Thesis, EE Department, March 1974, PB 231-365/AS

TR-125 Mark, W.S.
A Model-Debugging System, S.B. & S.M. Thesis, EE Department, April 1974; AD 778-688

TR-126 Altman, V.E.
A Language Implementation System, S.B. & S.M. Thesis, Sloan School, May 1974; AD 780-672

TR-127 Greenberg, B.S.
An Experimental Analysis of Program Reference Patterns in the Multics Virtual Memory, S.M. Thesis, EE Department, May 1974; AD 780-407

TR-128 Frankston, R.M.
The Computer Utility as a Marketplace for Computer Services, S.M. & E.E. Thesis, EE Department, May 1974; AD 780-436

TR-129 Weissberg, R.W.
Using Interactive Graphics in Simulating the Hospital Emergency Room, S.M. Thesis, EE Department May 1974; AD 780-437

TR-130 Ruth, G.R.
Analysis of Algorithm Implementations, Ph.D. Dissertation, EE Department, May 1974; AD 780-408

TR-131	Levin, M. Mathematical Logic for Computer Scientists, June 1974.
TR-132	Janson, P.A. Removing the Dynamic Linker from the Security Kernel of a Computing Utility, S.M. Thesis, EE Department, June 1974; AD 781-305
TR-133	Stockmeyer, L.J. The Complexity of Decision Problems in Automata Theory and Logic, Ph.D. Dissertation, EE Department, July 1974, PB 235-283/AS
TR-134	Ellis, D.J. Semantics of Data Structures and References, S.M. & E.E. Thesis, EE Department, August 1974, PB 236-594/AS
TR-135	Pfister, G.F. The Computer Control of Changing Pictures, Ph.D. Dissertation, EE Department, September 1974; AD 787-795
TR-136	Ward, S.A. Functional Domains of Applicative Languages, Ph.D. Dissertation, EE Department, September 1974; AD 787-796
TR-137	Seiferas, J.I. Nondeterministic Time and Space Complexity Classes, Ph.D. Dissertation, Math. Department, September 1974. PB 236-777/AS
TR-138	Yun, D.Y.Y. The Hensel Lemma in Algebraic Manipulation, Ph.D. Dissertation, Math. Department, November 1974; AD A002-737
TR-139	Ferrante, J. Some Upper and Lower Bounds on Decision Procedures in Logic, Ph.D. Dissertation, Math. Department, November 1974. PB 238-121/AS
TR-140	Redell, D.D. Naming and Protection in Extendable Operating Systems, Ph.D. Dissertation, EE Department, November 1974; AD A001-721
TR-141	Richards, M., Evans, A. and Mabee, R. The BCPL Reference Manual, December 1974; AD A003-599
TR-142	Brown, G.P. Some Problems in German to English Machine Translation, S.M. & E.E. Thesis, EE Department, December 1974; AD A003-002
TR-143	Silverman, H. A Digitalis Therapy Advisor, S.M. Thesis, EE Department, January 1975.
TR-144	Rackoff, C. The Computational Complexity of Some Logical Theories, Ph.D. Dissertation, EE Department, February 1975.

TR-145	Henderson, D.A. The Binding Model: A Semantic Base for Modular Programming Systems, Ph.D. Dissertation, EE Department, February 1975; AD A006-961
TR-146	Malhotra, A. Design Criteria for a Knowledge-Based English Language System for Management: An Experimental Analysis, Ph.D. Dissertation, EE Department, February 1975.
TR-147	Van De Vanter, M.L. A Formalization and Correctness Proof of the CGOL Language System, S.M. Thesis, EE Department, March 1975.
TR-148	Johnson, J. Program Restructuring for Virtual Memory Systems, Ph.D. Dissertation, EE Department, March 1975; AD A009-218
TR-149	Snyder, A. A Portable Compiler for the Language C, S.B. & S.M. Thesis, EE Department, May 1975; AD A010-218
TR-150	Rumbaugh, J.E. A Parallel Asynchronous Computer Architecture for Data Flow Programs, Ph.D. Dissertation, EE Department, May 1975; AD A010-918
TR-151	Manning, F.B. Automatic Test, Configuration, and Repair of Cellular Arrays, Ph.D. Dissertation, EE Department, June 1975; AD A012-822
TR-152	Qualitz, J.E. Equivalence Problems for Monadic Schemas, Ph.D. Dissertation, EE Department, June 1975; AD A012-823
TR-153	Miller, P.B. Strategy Selection in Medical Diagnosis, S.M. Thesis, EE & CS Department, September 1975.
TR-154	Greif, I. Semantics of Communicating Parallel Processes, Ph.D. Dissertation, EE & CS Department, September 1975; AD A016-302
TR-155	Kahn, K.M. Mechanization of Temporal Knowledge, S.M. Thesis, EE & CS Department, September 1975.
TR-156	Bratt, R.G. Minimizing the Naming Facilities Requiring Protection in a Computer Utility, S.M. Thesis, EE & CS Department, September 1975.
TR-157	Meldman, J.A. A Preliminary Study in Computer-Aided Legal Analysis, Ph.D. Dissertation, EE & CS Department, November 1975; AD A018-997

TR-158	Grossman, R.W. Some Data-base Applications of Constraint Expressions, S.M. Thesis, EE & CS Department, February 1976; AD A024-149
TR-159	Hack, M. Petri Net Languages, March 1976.
TR-160	Bosyj, M. A Program for the Design of Procurement Systems, S.M. Thesis, EE & CS Department, May 1976; AD A026-688
TR-161	Hack, M. Decidability Questions, Ph.D. Dissertation, EE & CS Department, June 1976.
TR-162	Kent, S.T. Encryption-Based Protection Protocols for Interactive User-Computer Communication, S.M. Thesis, EE & CS Department, June 1976; AD A026-911
TR-163	Montgomery, W.A. A Secure and Flexible Model of Process Initiation for a Computer Utility, S.M. & E.E. Thesis, EE & CS Department, June 1976.
TR-164	Reed, D.P. Processor Multiplexing in a Layered Operating System, S.M. Thesis, EE & CS Department, July 1976.
TR-165	McLeod, D.J. High Level Expression of Semantic Integrity Specifications in a Relational Data Base System, S.M. Thesis, EE & CS Department, September 1976; AD A034-184
TR-166	Chan, A.Y. Index Selection in a Self-Adaptive Relational Data Base Management System, S.M. Thesis, EE & CS Department, September 1976; AD A034-185
TR-167	Janson, P.A. Using Type Extension to Organize Virtual Memory Mechanisms, Ph.D. Dissertation, EE & CS Department, September 1976.
TR-168	Pratt, V.R. Semantical Considerations on Floyd-Hoare Logic, September 1976.
TR-169	Safran, C., Desforges, J.F. and Tsichlis, P.N. Diagnostic Planning and Cancer Management, September 1976.
TR-170	Furtek, F.C. The Logic of Systems, Ph.D. Dissertation, EE & CS Department, December 1976.
TR-171	Huber, A.R. A Multi-Process Design of a Paging System, S.M. & E.E. Thesis, EE & CS Department, December 1976.
TR-172	Mark, W.S.

The Reformulation Model of Expertise, Ph.D. Dissertation, EE & CS Department, December 1976; AD A035-397

TR-173 Goodman, N.
Coordination of Parallel Processes in the Actor Model of Computation, S.M. Thesis, EE & CS Department, December 1976.

TR-174 Hunt, D.H.
A Case Study of Intermodule Dependencies in a Virtual Memory Subsystem, S.M. & E.E. Thesis, EE & CS Department, December 1976.

TR-175 Goldberg, H.J.
A Robust Environment for Program Development, S.M. Thesis, EE & CS Department, February 1977.

TR-176 Swartout, W.R.
A Digitalis Therapy Advisor with Explanations, S.M. Thesis, EE & CS Department, February 1977.

TR-177 Mason, A.H.
A Layered Virtual Memory Manager, S.M. & E.E. Thesis, EE & CS Department, May 1977.

TR-178 Bishop, P.B.
Computer Systems with a Very Large Address Space and Garbage Collection, Ph.D. Dissertation, EE & CS Department, May 1977; AD A040-601

TR-179 Karger, P.A.
Non-Discretionary Access Control for Decentralized Computing Systems, S.M. Thesis, EE & CS Department, May 1977; AD A040-804

TR-180 Luniewski, A.W.
A Simple and Flexible System Initialization Mechanism, S.M. & E.E. Thesis, EE & CS Department, May 1977.

TR-181 Mayr, E.W.
The Complexity of the Finite Containment Problem for Petri Nets, S.M. Thesis, EE & CS Department, June 1977 .

TR-182 Brown, G.P.
A Framework for Processing Dialogue, June 1977; AD A042-370

TR-183 Jaffe, J.M.
Semilinear Sets and Applications, S.M. Thesis, EE & CS Department, July 1977.

TR-184 Levine, P.H.
Facilitating Interprocess Communication in a Heterogeneous Network Environment, S.B. & S.M. Thesis, EE & CS Department, July 1977; AD A043-901

TR-185 Goldman, B.
Deadlock Detection in Computer Networks, S.B. & S.M. Thesis, EE & CS Department, September 1977; AD A047-025

TR-186 Ackerman, W.B.
A Structure Memory for Data Flow Computers, S.M. Thesis, EE & CS Department, September 1977; AD A047-026

TR-187 Long, W.J.
A Program Writer, Ph.D. Dissertation, EE & CS Department, November 1977; AD A047-595

TR-188 Bryant, R.E.
Simulation of Packet Communication Architecture Computer Systems, S.M. Thesis, EE & CS Department, November 1977; AD A048-290

TR-189 Ellis, D.J.
Formal Specifications for Packet Communication Systems, Ph.D. Dissertation, EE & CS Department, November 1977; AD A048-980

TR-190 Moss, J.E.B.
Abstract Data Types in Stack Based Languages, S.M. Thesis, EE & CS Department, February 1978; AD A052-332

TR-191 Yonezawa, A.
Specification and Verification Techniques for Parallel Programs Based on Message Passing Semantics, Ph.D. Dissertation, EE & CS Department, January 1978; AD A051-149

TR-192 Niamir, B.
Attribute Partitioning in a Self-Adaptive Relational Database System, S.M. Thesis, EE & CS Department, January 1978; AD A053-292

TR-193 Schaffert, J.C.
A Formal Definition of CLU, S.M. Thesis, EE & CS Department, January 1978

TR-194 Hewitt, C. and Baker, H., Jr.
Actors and Continuous Functionals, February 1978; AD A052-266

TR-195 Bruss, A.R.
On Time-Space Classes and Their Relation to the Theory of Real Addition, S.M. Thesis, EE & CS Department, March 1978

TR-196 Schroeder, M.D., Clark, D.D., Saltzer, J.H. and Wells, D.H.
Final Report of the Multics Kernel Design Project, March 1978

TR-197 Baker, H., Jr.
Actor Systems for Real-Time Computation, Ph.D. Dissertation, EE & CS Department, March 1978; AD A053-328

TR-198 Halstead, R.H., Jr.
Multiple-Processor Implementation of Message-Passing Systems, S.M. Thesis, EE & CS Department, April 1978; AD A054-009

TR-199 Terman, C.J.
The Specification of Code Generation Algorithms, S.M. Thesis, EE & CS Department, April 1978; AD A054-301

TR-200 Harel, D.
Logics of Programs: Axiomatics and Descriptive Power, Ph.D. Dissertation, EE & CS Department, May 1978

TR-201 Scheifler, R.W.
A Denotational Semantics of CLU, S.M. Thesis, EE & CS Department, June 1978

TR-202 Principato, R.N., Jr.
A Formalization of the State Machine Specification Technique, S.M. & E.E. Thesis, EE & CS Department, July 1978

TR-203 Laventhal, M.S.
Synthesis of Synchronization Code for Data Abstractions, Ph.D. Dissertation, EE & CS Department, July 1978; AD A058-232

TR-204 Teixeira, T.J.
Real-Time Control Structures for Block Diagram Schemata, S.M. Thesis, EE & CS Department, August 1978; AD A061-122

TR-205 Reed, D.P.
Naming and Synchronization in a Decentralized Computer System, Ph.D. Dissertation, EE & CS Department, October 1978; AD A061-407

TR-206 Borkin, S.A.
Equivalence Properties of Semantic Data Models for Database Systems, Ph.D. Dissertation, EE & CS Department, January 1979; AD A066-386

TR-207 Montgomery, W.A.
Robust Concurrency Control for a Distributed Information System, Ph.D. Dissertation, EE & CS Department, January 1979; AD A066-996

TR-208 Krizan, B.C.
A Minicomputer Network Simulation System, S.B. & S.M. Thesis, EE & CS Department, February 1979

TR-209 Snyder, A.
A Machine Architecture to Support an Object-Oriented Language, Ph.D. Dissertation, EE & CS Department, March 1979; AD A068-111

TR-210 Papadimitriou, C.H.
Serializability of Concurrent Database Updates, March 1979

TR-211 Bloom, T.
Synchronization Mechanisms for Modular Programming Languages, S.M. Thesis, EE & CS Department, April 1979; AD A069-819

TR-212 Rabin, M.O.
Digitalized Signatures and Public-Key Functions as Intractable as Factorization, March 1979

TR-213 Rabin, M.O.
Probabilistic Algorithms in Finite Fields, March 1979

TR-214 McLeod, D.
A Semantic Data Base Model and Its Associated Structured User Interface, Ph.D. Dissertation, EE & CS Department, March 1979; AD A068-112

TR-215 Svobodova, L., Liskov, B. and Clark, D.
Distributed Computer Systems: Structure and Semantics, April 1979; AD A070-286

TR-216 Myers, J.M.
Analysis of the SIMPLE Code for Dataflow Computation, June 1979

TR-217 Brown, D.J.
Storage and Access Costs for Implementations of Variable - Length Lists, Ph.D. Dissertation, EE & CS Department, June 1979

TR-218 Ackerman, W.B. and Dennis, J.B.
VAL--A Value-Oriented Algorithmic Language: Preliminary Reference Manual, June 1979; AD A072-394

TR-219 Sollins, K.R.
Copying Complex Structures in a Distributed System, S.M. Thesis, EE & CS Department, July 1979; AD A072-441

TR-220 Kosinski, P.R.
Denotational Semantics of Determinate and Non-Determinate Data Flow Programs, Ph.D. Dissertation, EE & CS Department, July 1979

TR-221 Berzins, V.A.
Abstract Model Specifications for Data Abstractions, Ph.D. Dissertation, EE & CS Department, July 1979

TR-222 Halstead, R.H., Jr.
Reference Tree Networks: Virtual Machine and Implementation, Ph.D. Dissertation, EE & CS Department, September 1979; AD A076-570

TR-223 Brown, G.P.
Toward a Computational Theory of Indirect Speech Acts, October 1979; AD A077-065

TR-224 Isaman, D.L.
Data-Structuring Operations in Concurrent Computations, Ph.D. Dissertation, EE & CS Department, October 1979

TR-225 Liskov, B., Atkinson, R., Bloom, T., Moss, E., Schaffert, C., Scheifler, R. and Snyder, A.
CLU Reference Manual, October 1979; AD A077-018

TR-226 Reuveni, A.
The Event Based Language and Its Multiple Processor Implementations, Ph.D. Dissertation, EE & CS Department, January 1980; AD A081-950

TR-227	Rosenberg, R.L. Incomprehensible Computer Systems: Knowledge Without Wisdom, S.M. Thesis, EE & CS Department, January 1980
TR-228	Weng, K-S. An Abstract Implementation for a Generalized Data Flow Language, Ph.D. Dissertation, EE & CS Department, January 1980
TR-229	Atkinson, R.R. Automatic Verification of Serializers, Ph.D. Dissertation, EE & CS Department, March 1980; AD A082-885
TR-230	Baratz, A.E. The Complexity of the Maximum Network Flow Problem, S.M. Thesis, EE & CS Department, March 1980
TR-231	Jaffe, J.M. Parallel Computation: Synchronization, Scheduling, and Schemes, Ph.D. Dissertation, EE & CS Department, March 1980
TR-232	Luniewski, A.W. The Architecture of an Object Based Personal Computer, Ph.D. Dissertation, EE & CS Department, March 1980; AD A083-433
TR-233	Kaiser, G.E. Automatic Extension of an Augmented Transition Network Grammar for Morse Code Conversations, S.B. Thesis, EE & CS Department, April 1980; AD A084-411
TR-234	Herlihy, M.P. TRansmitting Abstract Values in Messages, S.M. Thesis, EE & CS Department, May 1980; AD A086-984
TR-235	Levin, L.A. A Concept of Indepandence with Applications in Various Fields of Mathematics, May 1980
TR-236	Lloyd, E.L. Scheduling Task Systems with Resources, Ph.D. Dissertation, EE & CS Department, May 1980
TR-237	Kapur, D. Towards a Theory for Abstract Data Types, Ph.D. Dissertation, EE & CS Department, June 1980; AD A085-877
TR-238	Bloniarz, P.A. The Complexity of Monotone Boolean Functions and an Algorithm for Finding Shortest Paths in a Graph, Ph.D. Dissertation, EE & CS Department, June 1980
TR-239	Baker, C.M. Artwork Analysis Tools for VLSI Circuits, S.M. & E.E. Thesis, EE & CS Department, June 1980; AD A087-040
TR-240	Montz, L.B. Safety and Optimization Transformations for Data Flow Programs, S.M. Thesis, EE & CS Department, July 1980

TR-241	Archer, R.F., Jr. Representation and Analysis of Real-Time Control Structures, S.M. Thesis, EE & CS Department, August 1980; AD A089-828
TR-242	Loui, M.C. Simulations Among Multidimensional Turing Machines, Ph.D. Dissertation, EE & CS Department, August 1980
TR-243	Svobodova, L. Management of Object Histories in the Swallow Repository, August 1980; AD A089-836
TR-244	Ruth, G.R. Data Driven Loops, August 1980
TR-245	Church, K.W. On Memory Limitations in Natural Language Processing, S.M. Thesis, EE & CS Department, September 1980
TR-246	Tiuryn, J. A Survey of the Logic of Effective Definitions, October 1980
TR-247	Weihl, W.E. Interprocedural Data Flow Analysis in the Presence of Pointers, Procedure Variables, and Label Variables, S.B.& S.M.Thesis, EE & CS Department, October 1980
TR-248	LaPaugh, A.S. Algorithms for Integrated Circuit Layout: An Analytic Approach, Ph.D.Dissertation, EE & CS Department, November 1980
TR-249	Turkle, S. Computers and People: Personal Computation, December 1980
TR-250	Leung, C.K.C. Fault Tolerance in Packet Communication Computer Architectures, Ph.D. Dissertation, EE & CS Department, December 1980
TR-251	Swartout, W.R. Producing Explanations and Justifications of Expert Consulting Programs, Ph.D. Dissertation, EE & CS Department, January 1981
TR-252	Arens, G.C. Recovery of the Swallow Repository, S.M. Thesis, EE & CS Department, January 1981; AD A096-374
TR-253	Ilson, R. An Integrated Approach to Formatted Document Production, S.M. Thesis, EE & CS Department, February 1981
TR-254	Ruth, G., Alter, S. and Martin, W. A Very High Level Language for Business Data Processing, March 1981
TR-255	Kent, S.T. Protecting Externally Supplied Software in Small Computers, Ph.D. Dissertation, EE & CS Department, March 1981

TR-256 Faust, G.G.
Semiautomatic Translation of COBOL into HIBOL, S.M. Thesis, EE & CS Department, April 1981

TR-257 Cisari, C.
Application of Data Flow Architecture to Computer Music Synthesis, S.B./S.M. Thesis, EE & CS Department, February 1981

TR-258 Singh, N.
A Design Methodology for Self-Timed Systems, S.M. Thesis, EE & CS Department, February 1981

TR-259 Bryant, R.E.
A Switch-Level Simulation Model for Integrated Logic Circuits, Ph.D. Dissertation, EE & CS Department, March 1981

TR-260 Moss, E.B.
Nested Transactions: An Approach to Reliable Distributed Computing, Ph.D. Dissertation, EE & CS Department, April 1981

TR-261 Martin, W.A., Church, K.W. and Patil, R.S.
Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results, EE & CS Department, June 1981

TR-262 Todd, K.W.
High Level Val Constructs in a Static Data Flow Machine, S.M. Thesis, EE & CS Department, June 1981

TR-263 Street, R.S.
Propositional Dynamic Logic of Looping and Converse, Ph.D. Dissertation, EE & CS Department, May 1981

TR-264 Schiffenbauer, R.D.
Interactive Debugging in a Distributed Computational Environment, S.M. Thesis, EE & CS Department, August 1981

TR-265 Thomas, R.E.
A Data Flow Architecture with Improved Asymptotic Performance, Ph.D. Dissertation, EE & CS Department, April 1981

TR-266 Good, M.
An Ease of Use Evaluation of an Integrated Editor and Formatter, S.M. Thesis, EE & CS Department, August 1981

TR-267 Patil, R.S.
Causal Representation of Patient Illness for Electrolyte and Acid-Base Diagnosis, Ph.D. Dissertation, EE & CS Department, October 1981

TR-268 Guttag, J.V., Kapur, D., Musser, D.R.
Derived Pairs, Overlap Closures, and Rewrite Dominoes: New Tools for Analyzing Term Rewriting Systems, EE & CS Department, December 1981

TR-269 Kanellakis, P.C.

The Complexity of Concurrency Control for Distributed Data Bases, Ph.D. Dissertation, EE & CS Department, December 1981

TR-270 Singh, V.
The Design of a Routing Service for Campus-Wide Internet Transport, S.M. Thesis, EE & CS Department, January 1982

TR-271 Rutherford, C.J., Davies, B., Barnett, A.I., Desforges, J.F.
A Computer System for Decision Analysis in Hodgkins Disease, EE & CS Department, February 1982

TR-272 Smith, B.C.
Reflection and Semantics in a Procedural Language, Ph.D. Dissertation, EE & CS Department, January 1982

TR-273 Estrin, D.L.
Data Communications via Cable Television Networks: Technical and Policy Considerations, S.M. Thesis, EE & CS Department, May 1982

TR-274 Leighton, F.T.
Layouts for the Shuffle-Exchange Graph and Lower Bound Techniques for VLSI, Ph.D. Dissertation, EE & CS Department, August 1981

TR-275 Kunin, J.S.
Analysis and Specification of Office Procedures, Ph.D. Dissertation, EE & CS Department, February 1982

TR-276 Srivas, M.K.
Automatic Synthesis of Implementations for Abstract Data Types from Algebraic Specifications, Ph.D. Dissertation, EE & CS Department, June 1982

TR-277 Johnson, M.G.
Efficient Modeling for Short Channel Mos Circuit Simulation, S.M. Thesis, EE & CS Department, August 1982

TR-278 Rosenstein, L.S.
Display Management in an Integrated Office, S.M. Thesis, EE & CS Department, January 1982

TR-279 Anderson, T.L.
The Design of a Multiprocessor Development System, S.M. Thesis, EE & CS Department, September 1982

TR-280 Guang-Rong, G.
An Implementation Scheme for Array Operations in Static Data Flow Computers, S.M. Thesis, EE & CS Department, May 1982

TR-281 Lynch, N.A.
Multilevel Atomicity - A New Correctness Criterion for Data Base Concurrency Control, EE & CS Department, August 1982

TR-282 Fischer, M.J., Lynch, N.A. and Paterson, M.S.
Impossibility of Distributed Consensus with One Faulty Process, EE & CS Department, September 1982

TR-283 Sherman, H.B.
A Comparative Study of Computer-Aided Clinical Diagnosis, S.M. Thesis, EE & CS Department, January 1981

TR-284 Cosmadakis, S.S.
Translating Updates of Relational Data Base Views, S.M. Thesis, EE & CS Department, February 1983

TR-285 Lynch, N.A.
Concurrency Control for Resilient Nested Transactions, EE & CS Department, February 1983

TR-286 Goree, J.A.
Internal Consistency of a Distributed Transaction System with Orphan Detection, S.M. Thesis, EE & CS Department, January 1983

TR-287 Bui, T.N.
On Bisecting Random Graphs, S.M. Thesis, EE & CS Department, March 1983

TR-288 Landau, S.E.
On Computing Galois Groups and its Application to Solvability by Radicals, Ph.D. Dissertation, EE & CS Department, March 1983

TR-289 Sirbu, M., Schoichet, S.R., Kunin, J.S., Hammer, M.M., Sutherland, J.B. and Zarmer, C.L.
Office Analysis: Methodology and Case Studies, EE & CS Department, March 1983

TR-290 Sutherland, J.B.
An Office Analysis and Diagnosis Methodology, S.M. Thesis, EE & CS Department, March 1983

TR-291 Pinter, R.Y.
The Impact of Layer Assignment Methods on Layout Algorithms for Integrated Circuits, Ph.D. Dissertation, EE & CS Department, August 1982

TR-292 Dornbrook, M. and Blank, M
The MDL Programming Language Primer, EE & CS Department, June 1980

TR-293 Galley, S.W. and Pfister, G.
The MDL Programming Language, EE & CS Department, May 1979

TR-294 Lebling, P.D.
The MDL Programming Environment, EE & CS Department, May 1980

TR-295 Pitman, K.M.
The Revised Maclisp Manual, EE & CS Department, June 1983

TR-296 Church, K.W.
Phrase-Structure Parsing: A Method for Taking Advantage of Allophonic Constraints, Ph.D. Dissertation, EE & CS Department, June 1983

TR-297	Mok, A.K. Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment, Ph.D. Dissertation, EE & CS Department, June 1983
TR-298	Krugler, K. Video Games and Computer Aided Instruction, EE & CS Department, June 1983
TR-299	Wing, J. A Two Tiered Approach to Specifying Programs, June 1983
TR-300	Cooper, G. An Argument for Soft Layering of Protocols, May 1983
TR-301	Valente, J.A. Creating a Computer-based Learning Environment for Physically Handicapped Children, Ph.D. Dissertation, EE & CS Department, September 1983
TR-302	Arvind, Dertouzos, M.L. and Iannucci, R.A. A Multiprocessor Emulation Facility, October 1983
TR-303	Bloom T. Dynamic Module Replacement in a Distributed Programming System, Ph.D. Dissertation, EE & CS Department, September 1983
TR-304	Terman, C.J. Simulation Tools for Digital LSI Design, Ph.D. Dissertation, EE & CS Department, September 1983
TR-305	Bhatt, S.N. and Leighton, F.T. A Framework for Solving VLSI Graph Layout Problems, Ph.D. Dissertation, EE & CS Department, October 1983
TR-306	Leung, K.C. and Lim, W. Y-P. PADL -- A Packet Architecture Description Language: A Preliminary Reference Manual, October 1983
TR-307	Guttag, J.V. and Horning, J.J. Preliminary Report on the Larch Shared Language, October 1983
TR-308	Oki, B.M. Reliable Object Storage to Support Atomic Actions, M.S. Thesis, EE & CS Department, November 1983
TR-309	Brock, J.D. A Formal Model of Non-determinate Dataflow Computation, Ph.D. Dissertation, EE & CS Department, November 1983
TR-310	Granville, R. Cohesion in Computer Text Generation: Lexical Substitution, M.S. Thesis, EE & CS Department, December 1983
TR-311	Burke, G.G., Carrette, G.J. and Eliot, C.R. NIL Reference Manual, M.S. Thesis, EE & CS Department, December 1983

TR-312	Landcaster, J. Naming in a Programming Support Environment, M.S. Thesis, EE & CS Department, April 1984
TR-313	Koile, K. The Design and Implementation of an Online Directory Assistance System, M.S. Thesis, EE & CS Department, April 1984
TR-314	Weihl, W. Specification and Implementation of Atomic Data Types, Ph.D. Dissertation, EE & CS Department, April 1984
TR-315	Coan, B. and Turpin, R. Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement, April 1984
TR-316	Comer, M.H. Loose Consistency in a Personal Computer Mail System, S.B. & M.S. Thesis, May 1984
TR-317	Traub, K.R. An Abstract Architecture for Parallel Graph Reduction, S.B. Thesis, May 1984
TR-324	Schooler, R. Partial Evaluation as a Means of Language Extensibility, S.M. Thesis/August 1984
TR-327	Chiu, S.Y. Debugging Distributed Computations in a Nested Atomic Action System, Ph.D. Dissertation/December 1984
TR-328	Carnese,D.J. Multiple Inheritance in Contemporary Programming Languages, September 1984
TR-329	Sacks, E. Qualitative Mathematical Reasoning, November 1984
TR-330	Sarin, S.K. Interactive On-Line Conferences, Ph.D. Dissertation/June 1984
TR-331	Sollins, K.R. Distributed Name Management, Ph.D. Dissertation/February 1985
TR-332	Culler, D.E. Resource Management for the Tagged Token Dataflow Architecture, S.M. Thesis/January 1985
TR-333	Arnold, J.M. Parallel Simulation of Digital LSI Circuits, S.M. Thesis/April 1984
TR-335	Lundelius, J. Synchronizing Clocks in a Distributed System, S.M. Thesis/August 1984
TR-336	Trilling, S.

Some Implications of Complexity Theory on Pseudo-Random Bit Generation, S.M. Thesis/January 1985

TR-337 Seiler, L.D.
A Hardware Assisted Methodology for VLSI Design Rule Checking, Ph.D. Dissertation/February 1985

TR-338 Koton, P.A.
Towards a Problem Solving System for Molecular Genetics, May 1985

TR-339 Soley, R.M.
Generic Software for Emulating Multiprocessor Architectures, A Thesis/ May 1985

TR-340 Wellman, M.P.
Reasoning About Preference Models, S.M. Thesis/May 1985

TR-341 Boughton, G.A.
Routing Networks for Packet Communication Systems, Ph.D. Dissertation/ August 1984

TR-342 Stark, E.W.
Foundations of a Theory of Specification for Distributed Systems, Ph.D. Dissertation/August 1984

TR-343 Forgaard, R.
A Program for Generating and Analyzing Term Rewriting Systems, S.M. Thesis/ September 1984

TR-344 Yelick, K.A.
A Generalized Approach to Equational Unification, S.M. Thesis/August 1985

TR-345 Estrin, D.L.
Access to Inter-organization Computer Networks, Ph.D. Dissertation/August 1985

TR-346 Cosmadakis, S.S.
Equational Theories and Database Constraints, Ph.D. Dissertation/August 1985

TR-347 Morecroft, L.E.
A Relative-motion Microworld, S.M. Thesis/September 1985

TR-348 Yedwab, L.
On Playing Well in a Sum of Games, S.M. Thesis/August 1985

TR-349 Eisenberg, M.A.
Bochser: An Integrated Scheme Programming System, S.M. Thesis/August 1985

TR-350 Seliger, R.
Design and Implementation of a Distributed Program for Collaborative Editing, S.M. Thesis/September 1985

TR-351 Bhatt, S.N.

	The Complexity of Graph Layout and Channel routing for VLSI, Ph.D. Dissertation, February 1984
TR-352	Lucassen, J.M., Gifford, D.K., Berlin, S.T. and Burmaster, D.E. Boston Community Information System User Manual (Version 6.0), April 1986
TR-353	Jagannathan, S. Data Backup and Recovery in a Computer Architecture for Functional Programming, S.M. Thesis/October 1985
TR-354	Stamos, J.W. Remote Evaluation, Ph.D. Dissertation/January 1986
TR-355	Guharoy, B. Data Structure Management in a Data Flow Computer System, S.M. Thesis/May 1985
TR-356	Beckerle, M.J. Logical Structures For Functional Languages, S.M.Thesis/February 1986
TR-357	Gibson, J.C. Computation Management in a Single Address Space System, M.S. Thesis/January 1986
TR-358	Chaing, C.J. Primitives for Real-Time Animation in Three Dimensions, S.M. Thesis/April 1986
TR-359	Feldmeier, D.C. A CATV-Based High-Speed Packet-Switching Network Design, S.M. Thesis/April 1986
TR-360	Kunstaetter, R. Intelligent Physiologic Modeling, S.M. Thesis/April 1986
TR-361	Barrington, D.A. Bounded Width Branching Programs, Ph.D. Dissertation/June 1986
TR-362	Kuszmaul, B.C. Simulating Applicative Architectures on the Connection Machine, M.S. Thesis/June 1986
TR-363	Marantz, J.D. Exploiting Parallelism in VLSI CAD, S.M. Thesis/June 1986
TR-364	Lynch, N., Blaustein, B. and Siegel, M. Correctness Conditions for Highly Available Replicated Databases, June 1986
TR-365	Morais, D.R. ID World: An Environment for the Development of Dataflow Programs Written in ID, May 1986
TR-366	Younis, S.G. The Clock Distribution System of the Multiprocessor Emulation Facility, S.B.Thesis/June 1986

Progress Reports

1. Project MAC Progress Report I, to July 1964; AD 465-088
2. Project MAC Progress Report II, July 1964-July 1965; AD 629-494
3. Project MAC Progress Report III, July 1965-July 1966; AD 648-346
4. Project Mac Progress Report IV, July 1966-July 1967; AD 681-342
5. Project MAC Progress Report V, July 1967-July 1968; AD 687-770
6. Project MAC Progress Report VI, July 1968-July 1969; AD 705-434
7. Project MAC Progress Report VII, July 1969-July 1970; AD 732-767
8. Project MAC Progress Report VIII, July 1970-July 1971; AD 735-148
9. Project MAC Progress Report IX, July 1971-July 1972; AD 756-689
10. Project MAC Progress Report X, July 1972-July 1973; AD 771-428
11. Project MAC Progress Report XI, July 1973-July 1974; AD A004-966
12. Laboratory for Computer Science Progress Report XII, July 1974-July 1975; AD A024-527
13. Laboratory for Computer Science Progress Report XIII, July 1975-July 1976; AD A061-246
14. Laboratory for Computer Science Progress Report XIV, July 1976-July 1977; AD A061-932
15. Laboratory for Computer Science Progress Report 15, July 1977-July 1978; AD A073-958
16. Laboratory for Computer Science Progress Report 16, July 1978-July 1979; AD A088-355
17. Laboratory for Computer Science Progress Report 17, July 1979-July 1980; AD A093-384
18. Laboratory for Computer Science Progress Report 18, July 1980-June 1981, A 127586
19. Laboratory for Computer Science Progress Report 19, July 1981-June 1982, A 143429
20. Laboratory for Computer Science Progress Report 20, July 1982-June 1983, A 145134
21. Laboratory for Computer Science Progress Report 21, July 1983-June 1984, A 154810
22. Laboratory for Computer Science Progress Report 22, July 1984-June 1985

Copies of all reports with A; AD, or PB numbers listed in Publications may be secured from the National Technical Information Service, U.S. Department of Commerce, Reports Division, 5285 Port Royal Road, Springfield, Virginia 22161 (tel: 703-487-4650). Prices vary. The reference number must be supplied with the request.

OFFICIAL DISTRIBUTION LIST

DIRECTOR Information Processing Techniques Office Defense Advanced Research Projects Agency (DARPA) 1400 Wilson Boulevard Arlington, VA 22209	2 copies
OFFICE OF NAVAL RESEARCH 800 North Quincy Street Arlington, VA 22217 Attn: Dr. Gary Koop, Code 433	2 copies
DIRECTOR, CODE 2627 Naval Research Laboratory Washington, DC 20375	6 copies
DEFENSE TECHNICAL INFORMATION CENTER Cameron Station Alexandria, VA 22314	12 copies
NATIONAL SCIENCE FOUNDATION Office of Computing Activities 1800 G. Street, N.W. Washington, DC 20550 Attn: Program Director	2 copies
HEAD, CODE 38 Research Department Naval Weapons Center China Lake, CA 93555	1 copy